LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-413

LEARNING BINARY RELATIONS AND TOTAL ORDERS

Sally A. Goldman
Ronald L. Rivest
Robert E. Schapire

May 1990

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TM-413	5. MONITORING ORGANIZATION REPORT NUMBER(S) N00014-89-J-1988
6a. NAME OF PERFORMING ORGANIZATION MIT Lab for Computer Science 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Dept. of Navy
6c. ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139	7b. ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) Learning Binary Relations and Total Orders	
12. PERSONAL AUTHOR(S) Sally Goldman, Ronald Rivest, Robert Schapire	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM ____ TO ____	14. DATE OF REPORT (Year, Month, Day) May 1990 15. PAGE COUNT 50
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB-GROUP	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Binary Relations, Total Orders, Computational Learning Theory, Machine Learning, Mistake-Bounded Learning, On-Line Learning, ~~Fully~~ Polynomial Randomized Approximation Schemes

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

We study the problem of learning a binary relation between two sets of objects or between a set and itself. We represent a binary relation between a set of size n and a set of size m as an $n \times m$ matrix of bits, whose (i,j) entry is 1 if and only if the relation holds between the corresponding elements of the two sets. ~~We present~~ polynomial prediction algorithms for learning binary relations in an extended on-line learning model, where the examples are drawn by the learner, by a helpful teacher, by an adversary, or according to a uniform probability distribution on the instance space.

In the first part of this paper, we present results for the case that the matrix of the relation has at most k row types. We present upper and lower bounds on the number of prediction mistakes any prediction algorithm makes when learning such a matrix under the extended on-line learning model. Furthermore, we describe a technique that simplifies the proof of expected mistake bounds against a randomly chosen query sequence.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Judy Little	22b. TELEPHONE (Include Area Code) (617) 253-5894 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

U.S. Government Printing Office: 1985—607-047

Unclassified

19. In the second part of this paper, we consider learning a binary relation that is a total order on a set. We describe a general technique using a fully polynomial randomized approximation scheme (fpras) to implement a randomized version of the halving algorithm. We apply this technique to the problem of learning a total order, using a fpras for counting the number of extensions of a partial order, to obtain a polynomial prediction algorithm that with high probability makes at most $n \lg n + (\lg e) \lg n$ mistakes when an adversary selects the query sequence. We also consider the case that a teacher or the learner selects the query sequence. Finally, we discuss how the halving algorithm may be used to construct an efficient counting algorithm.

Learning Binary Relations and Total Orders

Sally A. Goldman Ronald L. Rivest Robert E. Schapire

MIT Laboratory for Computer Science
Cambridge, Massachusetts 02139

May 4, 1990

Abstract

We study the problem of learning a binary relation between two sets of objects or between a set and itself. We represent a binary relation between a set of size n and a set of size m as an $n \times m$ matrix of bits, whose (i,j) entry is 1 if and only if the relation holds between the corresponding elements of the two sets. We present polynomial prediction algorithms for learning binary relations in an extended on-line learning model, where the examples are drawn by the learner, by a helpful teacher, by an adversary, or according to a uniform probability distribution on the instance space.

In the first part of this paper, we present results for the case that the matrix of the relation has at most k row types. We present upper and lower bounds on the number of prediction mistakes any prediction algorithm makes when learning such a matrix under the extended on-line learning model. Furthermore, we describe a technique that simplifies the proof of expected mistake bounds against a randomly chosen query sequence.

In the second part of this paper, we consider learning a binary relation that is a total order on a set. We describe a general technique using a fully polynomial randomized approximation scheme (fpras) to implement a randomized version of the halving algorithm. We apply this technique to the problem of learning a total order, using a fpras for counting the number of extensions of a partial order, to obtain a polynomial prediction algorithm that with high probability makes at most $n \lg n + (\lg e) \lg n$ mistakes when an adversary selects the query sequence. We also consider the case that a teacher or the learner selects the query sequence. Finally, we discuss how the halving algorithm may be used to construct an efficient counting algorithm.

Keywords: Binary Relations, Total Orders, Computational Learning Theory, Machine Learning, Mistake-Bounded Learning, On-Line Learning, Fully Polynomial Randomized Approximation Schemes

This paper prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, DARPA Contract N00014-89-J-1988, and a grant from the Siemens Corporation. A preliminary version of this paper will appear in the Proceedings of the 30th IEEE Symposium on Foundations of Computer Science.

Authors' network addresses: sally@theory.lcs.mit.edu, rivest@theory.lcs.mit.edu, rs@theory.lcs.mit.edu.

1 Introduction

In many domains, it is important to acquire information about a relation between two sets. For example, one may wish to learn a "has-part" relation between a set of animals and a set of attributes. We are motivated by the problem of designing a prediction algorithm to learn such a binary relation where the learner has limited prior information about the predicate forming the relation. While one could model such problems as concept learning, they are fundamentally different problems. In concept learning there is a single set of objects and the learner's task is to classify these objects, whereas in learning a binary relation there are two sets of objects and the learner's task is to learn the predicate relating the two sets. Observe that the problem of learning a binary relation can be viewed as a concept learning problem by letting the instances be all ordered pairs of objects from the two sets. However, the ways in which the problem may be structured are quite different when the true task is to learn a binary relation as opposed to a classification rule. That is, instead of a rule that defines which objects belong to the target concept, the predicate defines a relationship between pairs of object.

A binary relation is defined between two sets of objects. Throughout this paper, we assume that one set has cardinality n and the other has cardinality m. We also assume that for all possible pairings of objects, the predicate relating the two sets of variables is either true (1) or false (0). Before defining a prediction algorithm, we first discuss our representation of a binary relation. Throughout this paper, we represent the relation as an $n \times m$ binary matrix, where an entry contains the value of the predicate for the corresponding elements. Since the predicate is binary-valued, all entries in this matrix are either 0 (false) or 1 (true). The *two dimensional* structure arises from the fact that we are learning a binary relation.

For the sake of comparison, we now briefly mention other possible representations. One could represent the relation as a table with two columns, where each entry in the first column is an item from the first set and each entry in the second column is an item from the second set. The rows of the table consist of the subset of the potential nm pairings for which the predicate is true. One could also represent the relation as a bipartite graph with n vertices in one vertex set and m vertices in the other set. An edge is placed between two vertices exactly when the predicate is true for corresponding items.

Having introduced our method for representing the problem, we now informally discuss

the basic learning scenario. The learner is repeatedly given a pair of elements, one from each set, and asked to predict the corresponding matrix entry. After making its prediction, the learner is told the correct value of the matrix entry. The learner wishes to minimize the number of incorrect predictions it makes. Since we assume that the learner must eventually make a prediction for each matrix entry, the number of incorrect predictions depends on the size of the matrix.

Unlike problems typically studied where the natural measure of the size of the learner's problem is the size of an instance (or example), for this problem it is the size of the matrix. Such concept classes with polynomial-sized instance spaces are uninteresting in Valiant's [23] probably approximately correct (PAC) model of learning. In this model, instances are chosen randomly from an arbitrary unknown probability distribution on the instance space. A concept class is PAC-learnable if the learner, after seeing a number of instances that is polynomial in the problem size, can output a hypothesis that is correct on all but an arbitrarily small fraction of the instances with high probability. For concepts whose instance space has cardinality polynomial in the problem size, by asking to see enough instances the learner can see almost all of the probability weight of the instances space. Thus it is not hard to show that these concept classes are trivially PAC-learnable. One goal of our research is to build a framework for studying such problems.

To study learning algorithms for these concept classes we extend the basic mistake bound model [10, 11, 15] to the cases that a helpful teacher or the learner selects the query sequence, in addition to the cases where instances are chosen by an adversary or according to a probability distribution on the instance space. Previously, helpful teachers have been used to provide counterexamples to conjectured concepts [1], or to break up the concept into smaller sub-concepts [19]. In our framework, the teacher only selects the presentation order for the instances.

If the learner is to have any hope of doing better than random guessing, there must be some structure in the relation. Furthermore, since there are so many ways to structure a binary relation, we give the learner some prior knowledge about the nature of this structure. Not surprisingly, the learning task depends greatly on the prior knowledge provided. One way to impose structure is to restrict one set of objects to have relatively few "types." For example, a circus may contain many animals, but only a few different species. In the first part of this paper we study the case where the learner has "a priori" knowledge that there are

a limited number of object types. Namely, we restrict the matrix representing the relation to have at most k distinct row types. (Two rows are of the same type if they agree in all columns.) We define a *k-binary-relation* to be a binary relation for which the corresponding matrix has at most k row types. This restriction is satisfied whenever there are only k types of objects in the set of n objects being considered in the relation. The learner receives *no* other knowledge about the predicate forming the relation. With this restriction, we prove that any prediction algorithm makes at least $(1-\beta)km + n\lfloor \lg(\beta k)\rfloor - (1-\beta)k\lfloor \lg(\beta k)\rfloor$ mistakes in the worst case for all $0 < \beta \leq 1$ against any query sequence. So for $\beta = 1/2$, we get a lower bound of $\frac{km}{2} + (n - \frac{k}{2})\lfloor \lg k - 1\rfloor$ on the number of mistakes made by any prediction algorithm. If computational efficiency is not a concern, the halving algorithm [3, 15] makes at most $km + (n-k)\lg k$ mistakes against any query sequence. (The halving algorithm predicts according to the majority of the feasible relations (or concepts), and thus each mistake halves the number of remaining relations.)

We present an efficient algorithm making at most $km + (n-k)\lceil \lg k\rceil$ mistakes in the case that the learner chooses the query sequence. We prove a tight mistake bound of $km + (n-k)(k-1)$ in the case that the helpful teacher selects the query sequence [1]. When the adversary selects the query sequence, we present an efficient algorithm for $k = 2$ that makes at most $2m + n - 2$ mistakes, and for arbitrary k we present an efficient algorithm making at most $km + n\sqrt{(k-1)m}$ mistakes. We prove any algorithm makes at least $km + (n-k)\lfloor \lg k\rfloor$ mistakes in the case that an adversary selects the query sequence, and use the existence of projective geometries to improve this lower bound to $\Omega(km + (n-k)\lfloor \lg k\rfloor + \min\{n\sqrt{m}, m\sqrt{n}\})$ for a large class of algorithms. Finally, we describe a technique to simplify the proof of expected mistake bounds when the query sequence is chosen at random, and use it to prove an $O(km + nk\sqrt{H})$ expected mistake bound for a simple algorithm. (Here H is the maximum Hamming distance between any two rows.)

Another possibility for known structure is the problem of learning a binary relation on a set where the predicate induces a total order on the set. (For example the predicate may be "<".) In the second half of this paper we study the case in which the learner has "a priori" knowledge that the relation forms a total order. There we see that the *halving algorithm* [3, 15] yields a good mistake bound against any query sequence. This

[1] The mistake bound is a worst case mistake bound taken over all "consistent" learners. See Section 2 for formal definitions.

motivates a second goal of this research, to develop efficient implementations of the halving algorithm. We uncover an interesting application of randomized approximation schemes to computational learning theory. Namely, we describe a technique that uses a fully polynomial randomized approximation scheme (fpras) to implement a randomized version of the halving algorithm. We apply this technique, using a fpras due to Dyer, Frieze, and Kannan [7] and Matthews [18] for counting the number of linear extensions of a partial order, to obtain a polynomial prediction algorithm that makes at most $n \lg n + (\lg e) \lg n$ mistakes with very high probability against an adversary-selected query sequence. The small probability of making "too many" mistakes is determined by the coin flips of the learning algorithm and not by the query sequence selected by the adversary. We contrast this result with an $n - 1$ mistake bound when the learner selects the query sequence [25], and an $n - 1$ mistake bound when a teacher selects the query sequence.

Finally, we discuss the relationship between counting schemes and the halving algorithm. A *majority algorithm* takes as input a description of a set of objects, some of which are distinguished, and outputs a 1 if and only if at least half of the elements in the set are distinguished. On the other hand, a *counting algorithm* outputs the number of distinguished elements in the set. We present some preliminary results on using a majority algorithm to implement a counting algorithm.

The remainder of this paper is organized as follows. In the next section we formally introduce the basic problem, the learning scenario and the extended mistake bound model. In Section 3 we present our results for learning an k-binary-relation. After giving a motivating example in Section 3.1, in the next section we present some general mistake bounds. Then in Section 3.3 we consider when the learner selects the query sequence. Next we consider when a helpful teacher selects the query sequence. In Section 3.5 we consider when an adversary selects the query sequence. Finally, in Section 3.6 we consider when the query sequence is selected at random. In Section 4 we turn our attention to the problem of learning a total order. We begin by discussing the relationship between the halving algorithm and approximate counting schemes in Section 4.1. In particular, we describe how a fpras can be used to implement an approximate halving algorithm. Then in Section 4.2 we present our results on learning a total order. Finally, in Section 4.3, we describe techniques to convert majority algorithms to counting algorithms. We conclude with a summary of our results and discussion of open problems.

2 Learning Scenario and Mistake Bound Model

In this section we give formal definitions and discuss the learning scenario used in this paper. To be consistent with the literature, we shall discuss these models in terms of concept learning. As we have mentioned, the problem of learning a binary relation can be viewed in this framework by letting the instance be all pairs of objects, one from each of the two sets.

A *concept* c is a Boolean function on some domain of instances. A *concept class* C is a collection of concepts. The learner's goal is to infer some unknown target concept chosen from some *known* concept class. Often C is decomposed into subclasses C_n according to some natural dimension measure n. That is, for each $n \geq 1$, let X_n denote a finite *learning domain*. Let $X = \bigcup_{n \geq 1} X_n$, and $x \in X$ denote an *instance*. To illustrate these definitions, we consider the concept class of monomials. (A monomial is conjunction of literals, where each literal is either some Boolean variable or its negation.) For this concept class n is just the number of variables. Thus $|X_n| = 2^n$ where each $x \in X_n$ is chosen from $\{0,1\}^n$ and represents the assignment for each variable. For each $n \geq 1$, let $C_n \subseteq 2^{X_n}$ be a *family of concepts* on X_n. Let $C = \bigcup_{n \geq 1} C_n$ denote a *concept class* over X. For example if C_n contains monomials over n variables, then C is the class of all monomials. Given any concept $c \in C_n$, we say that x in c is a *positive instance* of c, and x in $X_n - c$ is a *negative instance* of c. In our example, the target concept for the class of monomials over five variables might be $x_1\overline{x_4}x_5$. Then the instance "10001" is a positive instance and "00001" is a negative instance. Finally, the *hypothesis space* of algorithm A is simply the set of all hypotheses (or rules) h that A may output. (A hypothesis for C_n must make a prediction for each $x \in X_n$.)

A *prediction algorithm* for C is an algorithm that runs under the following scenario. A *learning session* consists of a set of *trials*. In each trial, the learner is given given an unlabeled instance $x \in X_n$. The learner uses its current hypothesis to predict if x is a positive or negative instance of the target concept $c \in C_n$ and then the learner is told the correct classification of x. If the prediction was incorrect, the learner has made a *mistake*. Note that in this model there is no training phase. Instead, the learner receives *unlabeled instances* throughout the entire learning session. However, after each prediction the learner "discovers" the correct classification. This feedback can then be used by the learner to improve its hypothesis. A learner is *consistent* if, on every trial, there is some concept in C_n that agrees both with the learner's prediction and with all the labeled instances observed on

preceding trials.

The number of mistakes made by the learner depends on the sequence of instances presented. We extend the mistake bound model to include several methods for the selection of instances. A *query sequence* is a permutation $\pi = \langle x_1, x_2, \ldots, x_{|X_n|} \rangle$ of X_n where x_t is the instance presented to the learner at the t^{th} trial. We call the agent selecting the query sequence the *director*. We consider the following directors:

- **Learner** – The learner chooses π. To select x_t, the learner may use time polynomial in n and s, and all information obtained in the first $t-1$ trials. In this case we say that the learner is *self-directed*.

- **Helpful Teacher** – A teacher who knows the target concept and wants to minimize the learner's mistakes chooses π. To select x_t, the teacher uses knowledge of the target concept, $x_1, \ldots, x_{t-1}$, and the learner's predictions on $x_1, \ldots, x_{t-1}$. To avoid allowing the learner and teacher to have a coordinated strategy, in this scenario we consider the worst case mistake bound over all consistent learners. In the case we say the learner is *teacher-directed*.

- **Adversary** – The adversary who selected the target concept chooses π. This adversary, who tries to maximize the learner's mistakes, knows the learner's algorithm and has unlimited computing power. In this case we say the learner is *adversary-directed*.

- **Random** – In this model, π is selected randomly according to a uniform probability distribution on the permutations of X_n. Here the number of mistakes made by the learner for some target concept c in C_n is defined to be the expected number of mistakes over all possible query sequences. In this case we say the learner is *randomly-directed*.

We consider how a prediction algorithm's performance depends on the director. Let $\mathcal{A}^Z(\mathcal{C})$ denote the set of prediction algorithms for learning concept class C with director Z. For prediction algorithm $A \in \mathcal{A}^Z(\mathcal{C})$, we define the *mistake bound* $\mathrm{MB}(A, C_n)$ to be the worst case number of mistakes made by A for any target concept in C_n under any query sequence provided by Z. (When Z = adversary, $\mathrm{MB}(A, C_n) = M_A(C_n)$ as defined by Littlestone [15].) We say that A is a *polynomial prediction algorithm* if A makes each prediction in time polynomial in n.

3 Learning Binary Relations

In this section we applying the learning scenario of the extended mistake bound model to the concept class C of k-binary-relations. For this concept class the dimension measure is nm, the number of entries in the corresponding matrix, and $X_{nm} = \{1, \cdots, n\} \times \{1, \cdots, m\}$. An instance (i, j) is in the target concept $c \in C_{nm}$ if and only if the matrix entry in row i and column j is a 1. So in each trial the learner is repeatedly given an instance x from X_{nm} and asked to predict the corresponding matrix entry. After making its prediction, the learner is told the correct value of the matrix entry. The learner wishes to minimize the number of incorrect predictions it makes. Since we assume that the learner must eventually make a prediction for each matrix entry, the number of incorrect predictions depends on the size of the matrix.

We begin this section with a motivating example from the domain of allergy testing. We use this example to motivate both the restriction that the matrix has k row types and the use of the extended mistake bound model. We then present general upper and lower bounds on the number of mistakes made by the learner regardless of the director. Finally, we study the complexity of learning a k-binary-relation under each director.

3.1 Motivation: Allergist Example

In this section we use the following example taken from the domain of allergy testing to motivate the problem of learning a k-binary relation.

Consider an allergist with a set of patients to be tested for a given set of allergens. Each patient is either *highly allergic*, *mildly allergic*, or *not allergic* to any given allergen. The allergist may use either a *epicutaneous* (scratch) test in which the patient is given a fairly low dose of the allergen, or a *intradermal* (under the skin) test in which the patient is given a larger dose of the allergen. The patients reaction to the test is classified as *strong positive*, *weak positive* or *negative*. Figure 1 describes the reaction that occurs for each combination of allergy level and dosage level. Finally, we assume a strong positive reaction is extremely uncomfortable to the patient, but not dangerous.

What options does the allergist have in testing a patient for a given allergen? He could just perform the intradermal test (option 0). Another option (option 1) is to perform a epicutaneous test, and if it is not conclusive, then perform an intradermal test. (See Figure 2

	Epicutaneous (Scratch)	Intradermal (Under the Skin)
Not Allergic	negative	negative
Mildly Allergic	negative	weak positive
Highly Allergic	weak positive	strong positive

Figure 1: Summary of testing reactions for allergy testing example.

for decision trees describing these two testing options.) Which testing option is best? If the patient has no allergy or a mild allergy to the given allergen, then testing option 0 is best, since the patient need not return for the second test. However, if the patient is highly allergic to the given allergen, then testing option 1 is best, since the patient does not experience a bad reaction. We assume the inconvenience of going to the allergist twice is approximately the same as having a bad reaction. That is, the allergist has no preference to error in a particular direction. While the allergist's final goal is to determine each patient's allergies, we consider the problem of learning the optimal testing option for each combination of patient and allergen.

The allergist interacts with the environment as follows. In each "trial" the allergist is asked to predict the best testing option for a given patient/allergen pair. He is then told the testing results, thus learning whether the patient is not allergic, mildly allergic or highly allergic to the given aliergen. In other words, the allergist receives feedback as to the correct testing option. Note that we make no restrictions on how the hypothesis is represented as long as it can be evaluated in polynomial time. In other words, all we require is that given any patient/allergen pair, the allergist decides which test to perform in a "reasonable" amount of time.

How can the allergist possibly predict a patient's allergies? If the allergies of the patients are completely "random," then there is not much hope. What priori knowledge does the allergist have? He knows that people often have exactly the same allergies. So there are a set of "allergy types" that occur often. (We do not assume that the allergist has a priori knowledge of the actual allergy types.) This knowledge can help guide the allergist's

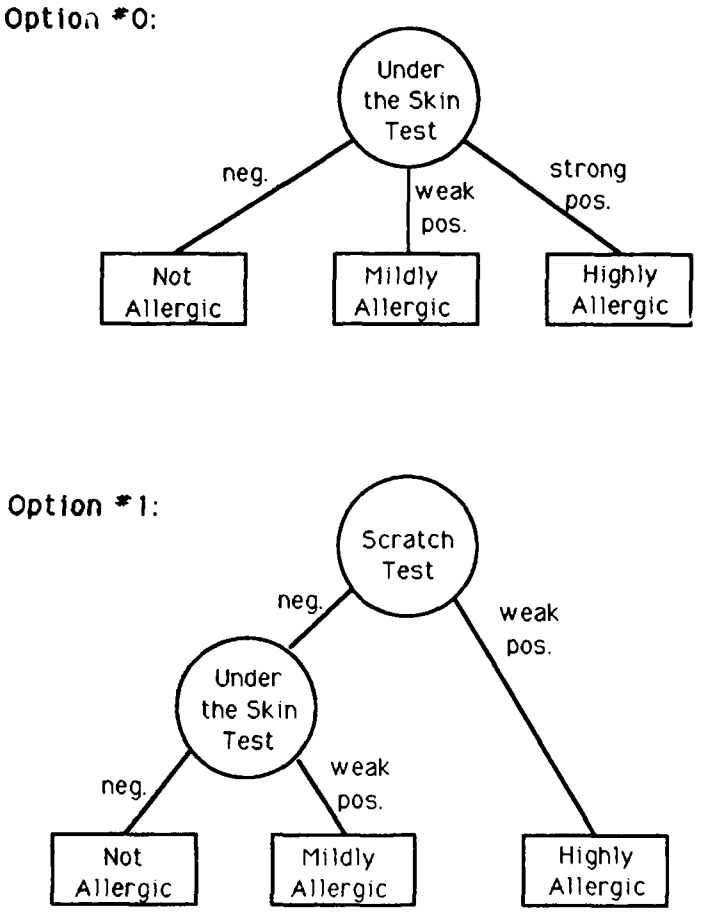

Figure 2: The testing options available to the allergist.

predictions.

Having specified the problem we discuss our choice of using the extended mistake bound model to evaluate learning algorithms for this problem. First of all, observe that we want an on-line model. There is no training phase here, the allergist wants to predict the correct testing option for each patient/allergen pair. Also we expect that the allergist has time to test each patient for each allergen, that is the instance space is polynomial-sized. Thus as discussed in Section 1 the distribution-free model is not appropriate.

How should we judge the performance of the learning algorithm? For each wrong prediction made, a patient is inconvenienced with making a second trip or having a bad reaction. Since the learner wants to give all patients the best possible service, he strives to minimize the number of incorrect predictions made. Thus we want to use the absolute mistake bound success criteria. Namely, we judge the performance of the learning algorithm by the number of incorrect predictions made during a learning session in which he must eventually test each patient for each allergen.

Up to now, the standard on-line model (using absolute mistake bounds to judge the learners) appears to be the appropriate model. We now discuss the selection of the instances. Since the allergist has no control over the target relation (*i.e. that allergies of his patients*), it makes sense to view the feedback as coming from an adversary. However, do we really want an adversary to select the presentation order for the instances? It could be that the allergist is working for a cosmetic company and due to restrictions of the Food and Drug Administration and the cosmetic company the allergist is essentially told when to test each person for each allergen. *In this case, it is appropriate to have an adversary select the* presentation order. However, in the typical situation, the allergist can decide in what order to perform the testing so that he can make the best predictions possible. In this case, we want to allow the learner to select the presentation order. One could also imagine the situation where a intern is being guided by an experienced allergist, and thus a teacher helps to select the presentation order. Finally, random selection of the presentation order may provide us with a better feeling for the behavior of an algorithm. Thus the learning model that is most appropriate for this example is the extended mistake bound model.

3.2 General Mistake Bounds

In this section we begin our study of learning k-binary-relations by presenting general lower and upper bounds on the mistakes made by the learner regardless of the director.

Throughout this section, we using the following notation. We say an entry (i, j) of the matrix (M_{ij}) is *known* if the learner was previously presented that entry. We assume without loss of generality that the learner is never asked to predict the value of a known entry. We say rows i and i' are *consistent* (given the current state of knowledge) if $M_{ij} = M_{i'j}$ for all columns j in which both entries i, j and i', j are known.

We now look at general lower and upper bounds on the number of mistakes that apply for all directors. Clearly, any learning algorithm makes at least km mistakes for some matrix, regardless of the query sequence. The adversary can divide the rows into k groups and reply that the prediction was incorrect for the first column queried for each entry of each group. We generalize this approach to force mistakes for more than one row of each type.

Theorem 1 *For any $0 < \beta \leq 1$, any prediction algorithm makes at least $(1-\beta)km + n\lfloor\lg(\beta k)\rfloor - (1-\beta)k\lfloor\lg(\beta k)\rfloor$ mistakes regardless of the query sequence.*

Proof: The adversary selects its feedback for the learner's predictions as follows. For each entry in the first $\lfloor\lg \beta k\rfloor$ columns the adversary replies that the learner's response is incorrect. At most βk new row types are created by this action. Likewise, for each entry in the first $(1-\beta)k$ rows the adversary replies that the learner's response is incorrect. This creates at most $(1-\beta)k$ new row types. The adversary makes all remaining entries in the matrix zero. (See Figure 3.) The number of mistakes is the area of the unmarked region. Thus the adversary has forced at least $(1-\beta)km + n\lfloor\lg(\beta k)\rfloor - (1-\beta)k\lfloor\lg(\beta k)\rfloor$ mistakes while creating at most $\beta k + (1-\beta)k = k$ row types. ∎

By letting $\beta = \frac{1}{2}$ we obtain the following corollary.

Corollary 1 *Any algorithm makes at least $\frac{km}{2} + (n - \frac{k}{2})\lfloor\lg k - 1\rfloor$ mistakes in the worst case regardless of the query sequence.*

If computational efficiency is not a concern, for all query sequences the halving algorithm [3, 15] provides a good mistake bound.

Observation 1 *The halving algorithm achieves a $km + n\lg k$ mistake bound.*

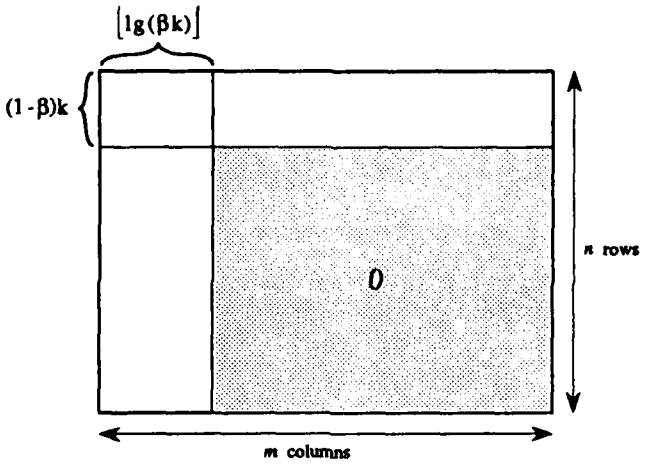

Figure 3: The final matrix created by the adversary in the proof of Theorem 1. All entries in the unmarked area will contain the bit not predicted by the learner. That is, a mistake is forced on each entry in the unmarked area. All entries in the marked area will be zero.

Proof: We use a simple counting argument on the size of the concept class. There are 2^{km} ways to select the k row types, and k^n ways to assign one of the k row types to each of the n rows. Thus $|C| \leq 2^{km}k^n$. Littlestone [15] proves that the halving algorithm makes at most $\lg|C|$ mistakes. thus the number of mistakes made by the halving algorithm for this concept class is at most $\lg(2^{km}k^n) \leq km + n\lg k$. ■

In the remainder of this section we study efficient prediction algorithms designed to perform well against each of the directors. For some directors we are also able to prove information-theoretic lower bounds that are better than that of Theorem 1. In Section 3.3 we consider the case that the query sequence is selected by the learner. We study the helpful-teacher director in Section 3.4. Finally, in Section 3.5 we consider the case of an adversary director.

3.3 Self-Directed Learning

In this section, we present an efficient algorithm for learning the matrix for the case in which the learner is the director.

Theorem 2 *There exists a polynomial prediction algorithm that achieves a* $km+(n-k)\lceil \lg k \rceil$ *mistake bound with a learner-selected query sequence.*

Proof: The query sequence selected simply specifies the entries of the matrix in row-major order. The learner begins assuming there is only one row type. Let $\hat{k}$ denote the learner's current estimate for k. So initially $\hat{k} = 1$. For the first row, the learner guesses each entry. (This row becomes the template for the first row type.) Next the learner assumes that the second row is the same as the first row. If he makes a mistake then the learner revises his estimate for $\hat{k}$ to be 2, guesses for the rest of the row, and uses that row as the template for the second row type. In general, to predict M_{ij}, the learner predicts according to a majority vote of the recorded row templates that are consistent with row i (breaking ties arbitrarily). Thus, if a mistake is made, then at least half of the row types can be eliminated as the potential type of row i. If more than $\lceil \lg \hat{k} \rceil$ mistakes are made in a row, then a new row type has been found. In this case, $\hat{k}$ is incremented, the learner guesses for the rest of the row, and makes this row the template for row type $\hat{k}+1$.

How many mistakes are made by this algorithm? Clearly, at most m mistakes are made for the first row found of each of the k types. For the remaining $n-k$ rows, since $\hat{k} \leq k$, at most $\lceil \lg k \rceil$ mistakes are made. ■

Note that this algorithm need not know k a priori. Furthermore, it obtains the same mistake bound even if an adversary tells the learner which row to examine, and in what order to predict the columns, provided that the learner sees all of a row before going on to the next. We note that this upper bound is within a constant factor of the lower bound of Corollary 1. However, this problem becomes harder if the adversary can select the query sequence without restriction.

3.4 Teacher-Directed Learning

In this section, we present upper and lower bounds on the number of mistakes made under the helpful-teacher director. Recall that in this model, we consider the worst case mistake bound over all consistent learners. Thus the question asked here is, what is the minimum number of matrix entries a teacher must reveal so that there is a unique completion of the matrix. That is, until there is a unique completion of the partial matrix, a mistake could be made on the next prediction.

We now prove an upper bound on the number of entries needed to uniquely define the target matrix.

Theorem 3 *The number of mistakes made with a helpful teacher as the director is at most* $km + (n-k)(k-1)$.

Proof: First, the teacher presents the learner with one row of each type. For each of the remaining $n-k$ rows the teacher presents an entry to distinguish the given row from each of the $k-1$ incorrect row types. After these $km + (n-k)(k-1)$ entries have been presented we claim that there is a unique matrix with at most k row types that is consistent with the partial matrix. Since all k distinct row types have been revealed in the first stage, all remaining rows must be the same as one of the first k rows presented. However, each of the remaining rows have been shown to be inconsistent with all but one of these k row templates. ■

Is Theorem 3 the best possible such result? Clearly the teacher must present a row of each type. But, in general, is it really necessary to present $k-1$ entries of the remaining rows to uniquely define the matrix? We now answer this question in the affirmative by presenting a matching lower bound.

Theorem 4 *The number of mistakes made with a helpful teacher as the director is at least* $\min\{nm, km + (n-k)(k-1)\}$.

Proof: The adversary selects the following matrix. The first row type consist of all zeros. For $2 \leq z \leq \min\{m+1, k\}$, rows type z contains $z-2$ zeros, followed by a one, followed by $m-z+1$ zeros. The first k rows are each assigned to be a different one of the k row types. Each remaining row is assigned to be the first row type. (See Figure 4.) Until there is a unique completion of the partial matrix, by definition there exists a consistent learner that could make a mistake. Clearly if the learner has not seen each column of each row type, then the final matrix is not uniquely defined. This part of the argument accounts for km mistakes. When $m+1 \geq k$, for the remaining rows unless all of the first $k-1$ columns are known, there is some row type besides the first row type that must be consistent with the given row. This argument accounts for $(n-k)(k-1)$ mistakes. Likewise, when $m+1 < k$ then if any of the first m columns are not known then there is some row type besides the first row type that must be consistent with the given row. This accounts for $(n-k)m$ mistakes. Thus the total number of mistakes is at least $\min\{nm, km + (n-k)(k-1)\}$. ■

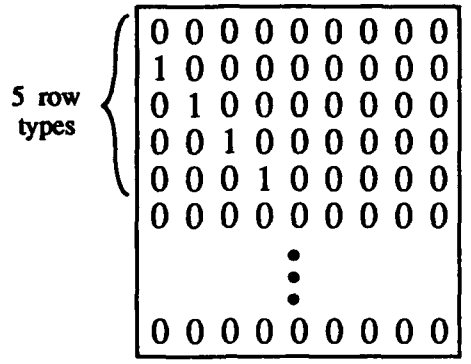

Figure 4: The matrix created by the adversary against the helpful teacher director. In this example, there are 5 row types which appear in the first five rows of the matrix.

We note that due to the restriction that the mistake bound in the helpful teacher director apply for all consistent learners, it is possible to get mistake bounds that are not as good as the bounds obtained when the learner is self-directed. Recall that in the previous section, we proved an $km + (n-k)\lceil \lg k \rceil$ mistake bound under the learner director. This bound is better than that obtained with a teacher because the learner uses a majority vote among the known row types for making predictions. However, a consistent learner may use a minority vote and could thus make $km + (n-k)(k-1)$ mistakes.

3.5 Adversary-Directed Learning

In this section we derive upper and lower bounds on the number of mistakes made by any learning algorithm for an adversary as the director. We first present an information-theoretic lower bound on the number of mistakes an adversary can force the learner to make. Next, we present an efficient prediction algorithm that achieves an optimal mistake bound if $k \leq 2$. Next we consider the related problem of computing the minimum number of row types needed to complete a partially known matrix. We then consider learning algorithms that work against an adversary for arbitrary k. We present an upper and lower bound on the number of mistakes made by a specific type of algorithm.

We now present an information-theoretic lower bound on the number of mistakes made by any prediction algorithm when the adversary selects the query sequence. We obtain this

result by modifying the technique used in Theorem 1.

Theorem 5 *Any prediction algorithm makes at least* $\min\{nm, km + (n-k)\lfloor \lg k \rfloor\}$ *mistakes against an adversary-selected query sequence.*

Proof: The adversary starts by presenting all entries in the first $\lfloor \lg k \rfloor$ columns (or m columns if $m < \lfloor \lg k \rfloor$) and replying that each prediction is incorrect. If $m \geq \lfloor \lg k \rfloor$, this step causes the learner to make $n\lfloor \lg k \rfloor$ mistakes. Otherwise, this step causes the learner to make nm mistakes. Each row can now be classified as one of k row types. Next the adversary presents the remaining columns for one row of each type, again replying that each prediction is incorrect. For $m \geq \lfloor \lg k \rfloor$ this step causes the learner to make $k(m - \lfloor \lg k \rfloor)$ additional mistakes. For the remaining matrix entries, the adversary replies as dictated by the completed row of the same row type as the given row. So the number of mistakes made by the learner is at least $\min\{nm, n\lfloor \lg k \rfloor + km - k\lfloor \lg k \rfloor\} = \min\{nm, km + (n-k)\lfloor \lg k \rfloor\}$. ■

3.5.1 Special Case: $k = 2$

We now consider efficient prediction algorithms for learning the matrix under an adversary-selected query sequence. (Recall that if efficiency is not a concern the halving algorithm makes at most $km + (n-k)\lg k$ mistakes.) In this section we consider the case that $k \leq 2$, and present an efficient prediction algorithm that performs optimally.

Theorem 6 *There exists a polynomial prediction algorithm that makes at most* $2m + n - 2$ *mistakes against an adversary-selected query sequence for* $k = 2$.

Proof: The algorithm uses a graph G whose vertices are the rows of the matrix and that initially has no edges. To predict M_{ij} the algorithm 2-colors the graph G, and then:

1. If no entry of column j is known, it guesses randomly.
2. Else if every known entry of column j is zero (respectively, one), it guesses zero (one).
3. Else it finds a row i' of the same color as i and known in column j, and guesses $M_{i'j}$.

Finally, after the prediction is made and the feedback received, the graph G is updated by adding an edge $\overline{ii'}$ to G for each row i' known in column j for which $M_{ij} \neq M_{i'j}$.Note that

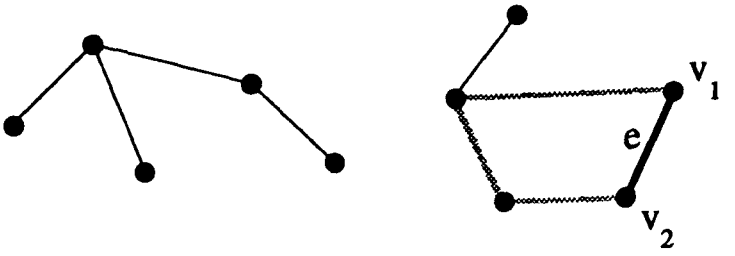

Figure 5: The situation occurring if G does not have a unique coloring after $n-1$ edges have been added. The thick grey edges and the thick black edge show the cycle in G. Let e (shown as a thick black edge) be the edge added to form the cycle.

one of the above cases always applies. Also, since $k = 2$, it will always be possible to find a 2-coloring.

How many mistakes can this algorithm make? It is not hard to see that cases 1 and 2 each occur only once for every column, so there are at most m mistakes made in each of these cases. Furthermore, the first case 2 mistake adds at least one edge to G. When a mistake is made in case 3, the algorithm learns that two rows are different, thus adding at least one edge to G between two nodes currently of the same color. So after at most $n-2$ case 3 mistakes, G has $n-1$ edges.

We now prove that when G has $n-1$ edges there is a unique 2-coloring[2] of G. Suppose there is not a unique 2-coloring. This assumption implies that there exists a set S of nodes, where S contains at least one node and at most $n-1$ nodes, such that the color of all nodes in S can be reversed without causing two adjacent nodes to be the same color. Thus it follows that G is not connected. However G has $n-1$ edges and thus there exists some connected component of G that must have a cycle. (See Figure 5.) We now separately consider the cases that this cycle contains an odd number of edges or an even number of edges.

- **Case 1: Odd-length cycle.** Let $e = \overline{v_1 v_2}$ be the last edge placed in the cycle. Consider the 2-coloring of G during the step of the algorithm when e was added. Since v_1 and v_2 were connected by an even number of edges, in any legal 2-coloring they must have been the same color. Thus we get a contradiction since a mistake could not

[2]Two colorings under renaming of the colors are considered to be the same.

have occurred.

- **Case 2: Even-length cycle.** Consider the last edge $e = \overline{v_1 v_2}$ placed in the cycle. Before e was added, since v_1 and v_2 were connected by an odd number of edges, in any legal 2-coloring they must have been different colors. Since Step 3 of the algorithm picks nodes of the same color, an edge could have never been placed between v_1 and v_2. Thus we again have a contradiction.

In both cases we reach a contradiction, and thus we have shown that after G has $n-1$ edges there is a unique 2-coloring. So after at most $n-2$ case 3 mistakes, there must be a unique coloring of G and no more mistakes can occur. Thus, the worst case number of mistakes made by this algorithm is $2m+n-2$. ■

Note that for $k = 2$ this upper bound matches the information-theoretic lower bound of Theorem 5. We also note that if there is only one row type then the algorithm given in Theorem 6 makes at most m mistakes, matching the information-theoretic lower bound.

An interesting theoretical question is to find a linear mistake bound for constant $k \geq 3$ when provided with a k-colorability oracle. However, such an approach would have to be greatly modified to yield a polynomial prediction algorithm since a polynomial-time k-colorability oracle exists only if $\mathcal{P} = \mathcal{NP}$. Furthermore, even good polynomial time approximations to a k-colorability oracle are not known [4, 14].

The remainder of this section focuses on designing polynomial prediction algorithms for the case that the matrix has at least three row types. One approach that may seem promising is to make predictions as follows. Compute a matrix that is consistent with all known entries and that has the fewest possible row types. Then use this matrix to make the next prediction. We now show that even computing the minimum number of row types needed to complete a partially known matrix is $\mathcal{NP}$-complete. Formally, we define the *matrix k-complexity* problem as follows: given an $n \times m$ binary matrix M that is partially known, decide if there is some matrix with at most k row types that is consistent with M. The matrix k-complexity problem can be shown to be $\mathcal{NP}$-complete by a reduction from graph k-colorability for the cases where $k > 2$ and $m \geq n$.

Theorem 7 *For $k > 2$ and $m \geq n$, the matrix k-complexity problem is $\mathcal{NP}$-complete.*

Proof: We use a reduction from graph k-colorability. Given an instance $G = (V, E)$ of graph k-colorability we transform it into an instance of the matrix k-complexity problem.

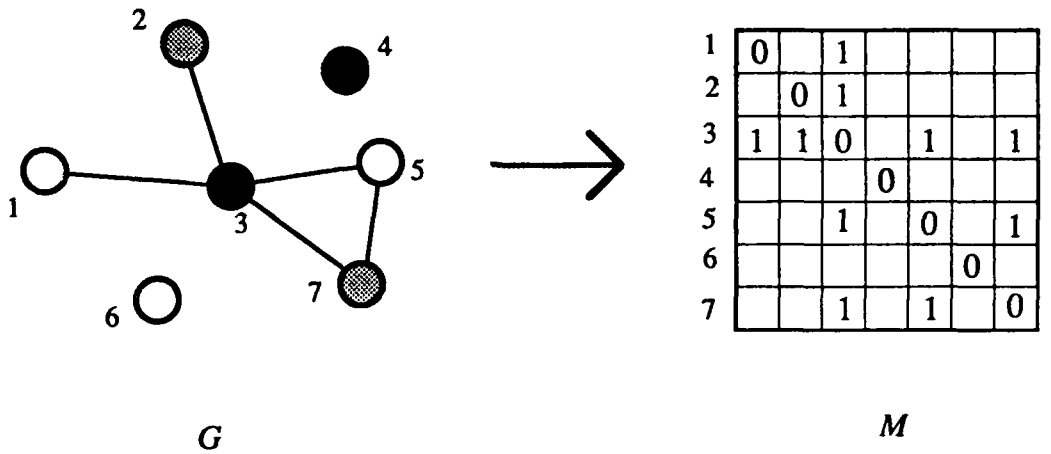

Figure 6: An example of the reduction used in Theorem 7. The graph G is the instance for the graph coloring problem. The partial matrix M is the instance for the matrix complexity problem. We note that there exists a matrix that is an completion of M that uses only three row types. The corresponding 3-coloring of G is demonstrated by the node colorings used in G.

Let $n = |V|$. For each edge $\{v_i, v_j\} \in E$, we add entries to the matrix so that row i and row j cannot be the same row type. Specifically, for each vertex v_i, we set $M_{ii} = 0$, and $M_{ji} = 1$ for each neighbor v_j of v_i. An example demonstrating this reduction is given in Figure 6.

We now show that there is some matrix of at most k row types that is consistent with this partial matrix if and only if G is k-colorable. We first argue that if there is a matrix M' consistent with M that has at most k row types then G is k-colorable. By the construction if two rows are of the same type there cannot be an edge between the corresponding nodes. So just let the node color for each node be the type of the corresponding row in M'.

We now argue that if G is k-colorable, then there exists a matrix M' consistent with M that has at most k row types. By the construction of M, if a set of vertices are the same color in G then the corresponding rows are consistent with each other. Thus there exists a matrix with at most k row types that is consistent with M. ∎

3.5.2 Row-Filter Algorithms

In this section we study the performance of a whole class of algorithms designed to learn a matrix with arbitrary complexity k when an adversary selects the query sequence. We say that an algorithm A is a *row-filter algorithm* if A makes its prediction for M_{ij} strictly as a function of j and all entries in the set I of rows consistent with row i and defined in column j. That is, A's prediction is $f(I, j)$ where f is some (possibly probabilistic) function. So, to make a prediction for M_{ij}, a row-filter algorithm considers all rows that could be the same type as row i and whose value for column j is known, and uses these rows in any way one could imagine to make a prediction. For example it could take a majority vote on the entries in column j of all rows that are consistent with row i. Or, of the rows defined in column j, it could select the row that has the most known values in common with row i and predict according to its entry in column j. We have found that many of the prediction algorithms we considered are row-filter algorithms.

Consider the simple row-filter algorithm, *ConsMajorityPredict*, in which $f(I, j)$ computes the majority vote of the entries in column j of the rows in I. (Guess randomly in the case of a tie.) Note that *ConsMajorityPredict* only takes time linear in the number of known entries of the matrix to make a prediction. We now give an upper bound on the number of mistakes made by *ConsMajorityPredict*.

Theorem 8 *The algorithm ConsMajorityPredict makes at most* $km + n\sqrt{(k-1)m}$ *mistakes against an adversary-selected query sequence.*

Proof: For all i, let $d(i)$ be the number of rows consistent with row i. We define the *potential* of a partially known matrix to be $\Phi = \sum_{i=1}^{n} d(i)$. We begin by considering how much the potential function can change over the entire learning session.

Lemma 1 *The potential function* Φ *decreases by at most* $\frac{k-1}{k}n^2$ *during the learning session.*

Proof: Initially for all i, $d(i) = n$. So $\Phi_{\text{init}} = n^2$. Let $C(z)$ be the number of rows of type z for $1 \leq z \leq k$. By definition, $\Phi_{\text{final}} = \sum_{z=1}^{k} C(z)^2$. Thus our goal is to minimize $\sum_{z=1}^{k} C(z)^2$ under the constraint that $\sum_{z=1}^{k} C(z) = n$. Using the method of Lagrange multipliers we obtain that Φ_{final} is minimized when for all z, $C(z) = n/k$. Thus $\Phi_{\text{final}} \geq (n/k)^2 k = n^2/k$. So $\Delta\Phi = \Phi_{\text{init}} - \Phi_{\text{final}} \leq n^2 - \frac{n^2}{k} = \frac{k-1}{k}n^2$. ∎

Now that the total decrease in Φ over the learning session is bounded, we need to determine how many mistakes can be made without Φ decreasing by more than $\frac{k-1}{k}n^2$. We begin

by noting that Φ is strictly decreasing. Once two rows are found to be inconsistent, they remain inconsistent. So for each i, $d(i)$ is strictly decreasing, and thus Φ is strictly decreasing. So to bound the number of mistakes made by *ConsMajorityPredict* we must compute a lower bound on the amount Φ is decreased by each mistake. Intuitively, one expects Φ to decrease by larger amounts as more of the matrix is seen. We formalize this intuition in the next two lemmas.

Lemma 2 *The r^{th} mistake made when predicting an entry M_{ij} in column j of some row i of type z decreases Φ by at least $2(r-1)$.*

Proof: Consider all the rows of type z. Since $r-1$ mistakes have occurred in column j of these rows, at least $r-1$ entries are known in column j of rows of type z. Since these rows must be in I, if a mistake occurs there must be at least $r-1$ entries in I (and thus consistent with row i) that differ in column j with row i. Thus if a mistake is made, row i is found to be inconsistent with at least $r-1$ rows it was thought to be consistent with. When two previously consistent rows are found to be inconsistent, Φ decreases by two. Thus the total decrease in Φ caused by the r^{th} mistake made when predicting an entry in column j of some row of type z decreases Φ by at least $2(r-1)$. ■

From Lemma 1, we see that the more entries that are known in a given column of a given row type, the greater the decrease in Φ for future mistakes on such entries. So, intuitively it appears that the adversary can maximize the number of mistakes made by the learner by balancing the number of entries seen for each column of each row type. We prove that this intuition is correct and apply it to obtain a lower bound on the amount Φ must have decreased after the learner has made μ mistakes.

Lemma 3 *After μ mistakes are made, the total decrease in Φ is at least $km\left(\frac{\mu}{km}-1\right)^2$.*

Proof: From Lemma 2, after the r^{th} mistake in column j of row type z, the total decrease in Φ from its initial value is at least $\sum_{x=1}^{r} 2(x-1) \geq (r-1)^2$. Let $W(j,z)$ be the number of mistakes made in column j of rows of type z. The total decrease in Φ is at least

$$D = \sum_{j=1}^{m}\sum_{z=1}^{k}(W(j,z)-1)^2$$

subject to the constraint $\sum_{j=1}^{m}\sum_{z=1}^{k} W(j,z) = \mu$.

Using the method of Lagrange multipliers, we obtain that d is minimized when $W(j,z) = \frac{\mu}{km}$ for all j and z. So the total decrease in Φ is at least

$$\sum_{j=1}^{m}\sum_{z=1}^{k}\left(\frac{\mu}{km}-1\right)^2 = km\left(\frac{\mu}{km}-1\right)^2.$$

■

We now complete the proof of the theorem. Combining Lemma 1 and Lemma 3 along with the observation that Φ is strictly non-increasing, we have shown that when

$$km\left(\frac{\mu}{km}-1\right)^2 \geq \frac{k-1}{k}n^2$$

then Φ must have decreased as much as it can and thus no more mistakes will occur. Solving for μ we obtain that Φ has decreased by its maximum amount when $\mu = km + n\sqrt{(k-1)m}$.

■

We note that by using the simpler argument that each mistake, except for the first mistake in each column of each row type, decreases Φ by at least 2, we obtain a $km + \frac{k-1}{2k}n^2$ mistake bound for *any* row-filter algorithm. Also, Manfred Warmuth [24] has independently given an algorithm, based on the weighted majority algorithm of Littlestone and Warmuth [16], that achieves an $O(km + n\sqrt{m \lg k})$ mistake bound. Warmuth's algorithm builds a complete graph of n vertices where row i corresponds to vertex v_i and all edges get an initial weight of 1. To predict a value for (i,j) the learner takes a weighted majority of all active neighbors of v_i (v_k is *active* if (k,j) is known). After receiving feedback, the learner sets the weight on the edge from v_i to v_k to be 0 if $(k,j) \neq (i,j)$. Finally, if a mistake occurs the learner doubles the weight of (v_i, v_k) if $(k,j) = (i,j)$ (i.e. the edges to neighbors that predicted correctly). We note that this algorithm is not a row-filter algorithm.

Does *ConsMajorityPredict* give the best performance possible by a row-filter algorithm? We now present an information-theoretic lower bound on the number of mistakes an adversary can force against any row-filter algorithm.

Theorem 9 *Let p be a prime and let $m = (p^2+p+1)$. Any row-filter algorithm for learning a $2n \times m$ matrix with $m \geq n$ and $k \geq 2$ makes at least $n(p+1) = \Omega(n\sqrt{m})$ mistakes when the adversary selects the query sequence.*

Proof: We assume that the adversary knows the learner's algorithm and has access to any random bits he uses. (One can prove a similar lower bound on the expected mistake bound when the adversary cannot access the random bits.)

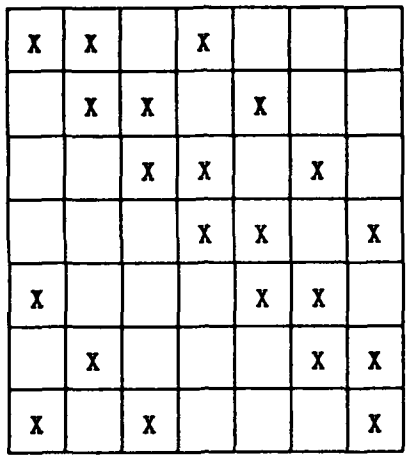

Figure 7: A projective geometry for $p = 2$, $m = 7$.

Our proof depends upon the existence of a projective geometry Γ on m points and lines [5]. That is, there exists a set of m points and a set of m lines such that each line contains exactly $p+1$ points and each point is at the intersection of exactly $p+1$ lines. Furthermore, any pair of lines intersects at exactly one point, and any two points define exactly one line. Figure 7 shows a matrix representation of such a geometry; an "x" in entry i, j indicates that point j is on line i. We use the first n lines of Γ.

The matrix M consists of two row types: the odd rows are filled with ones and the even rows with zeros. Imagine assigning two rows of M to each line of Γ. (See Figure 8). We now prove that the adversary can force a mistake for each entry of Γ. The adversary's query sequence maintains the condition that an entry i, j is not revealed unless line $\lceil i/2 \rceil$ of Γ contains point j. In particular, the adversary will begin by presenting one entry of the matrix for each entry of Γ. We prove that for each entry of Γ the learner must predict the same value for the two corresponding entries of the matrix. Thus the adversary forces a mistake for the $n(p+1) = \Omega(n\sqrt{m})$ entries of Γ. The remaining entries of the matrix are then presented in any order.

Let I be the set of rows that may be used be the row-filter algorithm when predicting entry $(2i, j)$. Let I' be the set of rows that may be used by the row-filter algorithm when predicting entry $(2i-1, j)$. We prove by contradiction that $I = I'$. If $I \neq I'$ then it must be the case that there is some row r that is defined in column j and consistent with row $2i$, yet inconsistent with row $2i-1$. By definition of the adversary's query sequence it must be

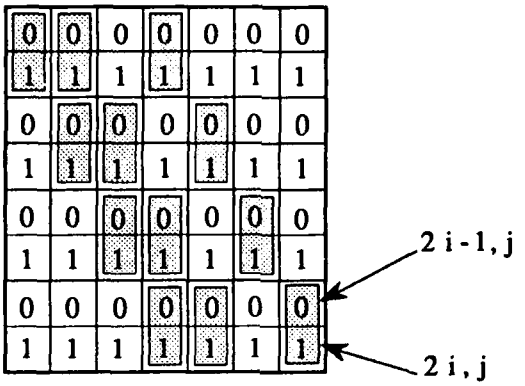

Figure 8: The matrix created by the adversary in the proof of Theorem 9. The shaded regions correspond to the entries in the first n lines of Γ. The learner is forced to make a mistake on one of the entries in each shaded rectangle.

the case that lines $\lceil r/2 \rceil$ and $\lceil (2i-1)/2 \rceil$ of Γ contain point j. Furthermore, since $2i-1, j$ is being queried that entry is not known. Thus rows r and $2i-1$ must both be known in *some other column j' since they* are known to be inconsistent. Thus since only entries in Γ are shown, it follows that lines $\lceil r/2 \rceil$ and $\lceil (2i-1)/2 \rceil$ of Γ also contain point j' for $j' \neq j$. So, this implies that lines $\lceil r/2 \rceil$ and $\lceil (2i-1)/2 \rceil$ of Γ must intersect at two points giving a contradiction. Thus $I = I'$ and so $f(I,j) = f(I',j)$ for entry $(2i,j)$ and entry $(2i-1,j)$. Since the rows differ in each column the adversary can force a mistake on one of these entries. Since the adversary has access to the random bits of the learner, he can compute $f(I,j)$ just before making his query, and ask the learner to predict the entry for which the mistake will be made. This procedure is repeated for the pair of entries corresponding to each element of Γ. ■

We use a similar argument to get an $\Omega(m\sqrt{n})$ bound for $m < n$. Combined with the lower bound of Theorem 5 and Theorem 9 we obtain a $\Omega(km + (n-k)\lfloor \lg k \rfloor + \min\{n\sqrt{m}, m\sqrt{n}\})$ lower bound on the number of mistakes made by a row-filter algorithm.

Corollary 2 *Any row-filter algorithm makes $\Omega(km + (n-k)\lfloor \lg k \rfloor + \min\{n\sqrt{m}, m\sqrt{n}\})$ mistakes against an adversary-selected query sequence.*

Comparing this lower bound to the upper bound proven for *ConsMajorityPredict*, we see that for fixed k the mistake bound of *ConsMajorityPredict* is within a constant factor of

optimal.

Given this lower bound, one may question the $2m+n-2$ upper bound for $k=2$ given in Theorem 6. However, the algorithm described is not a row-filter algorithm. Also compared to our results for the learner-selected query sequence, it appears that allowing the learner to select the query sequence is quite helpful.

3.6 Randomly-Directed Learning

In this section we consider the case that the learner is presented at each step with one of the remaining entries of the matrix selected uniformly and independently at random. We present a prediction algorithm that makes $O(km+nk\sqrt{H})$ mistakes on average where H is the maximum Hamming distance between any two rows of the matrix. We note that when $H=\Omega(\frac{m}{k})$ the result of Theorem 8 supersedes this result. A key result of this section is a proof relating two different probabilistic models for analyzing the mistake bounds under a random presentation. We first consider a simple probabilistic model in which the requirement that t matrix entries is known is simulated by assuming that each entry of the matrix is seen independently with probability $\frac{t}{nm}$. We then prove that any upper bound obtained on the number of mistakes under this simple probabilistic model holds under the true model (to within a constant factor) in which we have exactly t entries known. This result is extremely useful since in the true model the dependencies among the probabilities that matrix entries are known makes the prove significantly more difficult.

We define the algorithm *RandomConsistentPredict* to be the row-filter algorithm where the learner makes his prediction for M_{ij} by choosing one row i' of I uniformly at random and predicting the value of $M_{i'j}$. (If I is empty then *RandomConsistentPredict* makes a random guess.)

Theorem 10 *Let H be the maximum Hamming distance between any two rows of M. Then the expected number of mistakes made by RandomConsistentPredict is $O(k(n\sqrt{H}+m))$.*

Proof: Let U_t be the probability that the prediction rule makes a mistake on the $(t+1)$st step. That is, U_t is the chance that a prediction error occurs on the next randomly selected entry given that exactly t other randomly chosen entries are already known. Clearly, the expected number of mistakes is $\sum_{t=0}^{S-1} U_t$, where $S=nm$. Our goal is to find an upper bound for this sum.

The condition that exactly t entries are known makes the computation of U_t rather messy since the probability of having seen some entry of the matrix is *not* independent of knowing the others. Instead, we compute the probability V_t of a mistake under the simpler assumption that each entry of the matrix has been seen with probability t/S, independent of the rest of the matrix. We first compute an upper bound for the sum $\sum_{t=0}^{S-1} V_t$, and then show that this sum is within a constant factor of $\sum_{t=0}^{S-1} U_t$.

Lemma 4 $\sum_{t=0}^{S-1} V_t = O(km + nk\sqrt{H})$.

Proof: Fix t, and let $p = t/S$. For any row i we define $d(i)$ to be the number of rows of the same type as row i. We first prove, that without loss of generality, we can assume that $d(i) > 1$ for all $1 \leq i \leq n$. Namely, if this lemma holds for this restricted case, then it holds in the general case. Suppose there are n' rows that are each different from all other rows. Applying this lemma to the relation obtained by removing the n' objects that have distinct row types and then adding in the mistakes for these removed objects we get:

$$\begin{aligned}\sum_{t=0}^{S-1} V_t &= O\left((k-n')m + n'(k-n')\sqrt{H}\right) + n'm \\ &= O(km + n'(k-n')\sqrt{H}) \\ &= O(km + nk\sqrt{H}).\end{aligned}$$

Thus for the remainder of this proof we may assume that $d(i) > 1$ for all i.

We compute V_t as

$$V_t = \frac{1}{S}\sum_{i,j} \Pr[\text{a mistake is made} \mid M_{i,j} \text{ is unknown and } i,j \text{ is presented next}]. \quad (1)$$

If some of the rows known in column j are consistent with row i, then the probability of a mistake in row i, column j is the chance that one of these rows i' (chosen at random) differs from row i in the jth column, or

$$\frac{\sum_{i' \neq i} \Pr[M_{ij} \neq M_{i'j} \wedge \text{ rows } i, i' \text{ are consistent}]}{\sum_{i' \neq i} \Pr[\text{rows } i, i' \text{ are consistent}]}. \quad (2)$$

(For brevity, we omit the implicit conditions that entry i,j is unknown and i',j is known.)

We first focus on the numerator of Expression 2. We define r_0 as the expected number of rows of the same type as row i and defined in column j. Since in the $d(i) - 1$ rows of the same type as row i, column j is known with probability p,

$$r_0 = (d(i) - 1)\, p.$$

Next we define r_1 to be the expected number of rows of a different type from row i that are consistent with row i and defined in column j. Consider a column, c, in which some row i' differs from row i. With probability $1 - p^2$ either i or i' is not seen in column c and thus row i and i' are not detected to be inconsistent in column c. We know that row i is not known in column j and thus the number of columns for which we may detect that i and i' are different depends on whether i and i' differ in column j. Let $h(i, i')$ be the Hamming distance between rows i and i'. Let $h'(i, i')$ be the number of column in which i and i' differ excluding column j. Since column j of row i' is seen with probability p,

$$r_1 = \sum_{i' \neq i} p(1 - p^2)^{h'(i,i')}.$$

Finally, since $(1 - p^2) \leq 1$ and $h'(i, i') \geq h(i, i') - 1$,

$$r_1 \leq \sum_{i' \neq i} p(1 - p^2)^{h(i,i')-1}.$$

Finally, we define r_2 to be the expected number of rows that are consistent with row i and defined in column j, but different from row i in column j. Only the rows meeting the conditions for r_1 are considered in the set I, and with probability of $\frac{h(i,i')}{m}$, the row i' differs from row i in column j. Thus

$$r_2 \leq \sum_{i' \neq i} \frac{h(i, i')}{m} p(1 - p^2)^{h(i,i')-1}.$$

We can now evaluate the numerator of Expression (2). Namely,

$$\sum_{i' \neq i} \Pr[M_{ij} \neq M_{i'j} \wedge \text{ rows } i, i' \text{ are consistent}] = \frac{r_2}{r_0 + r_1} \leq \frac{r_2}{r_0}.$$

We now compute a lower bound on the denominator of Equation 2 so that we can get an upper bound on the entire expression. Since two rows of the same row type are always consistent, it follows that

$$\sum_{i' \neq i} \Pr[\text{rows } i, i' \text{ are consistent}] \geq d(i) - 1.$$

Recall Expression (2) for the probability of making a mistake when predicting M_{ij} given that some row consistent with row i was defined in column j. An error can also occur when there are no consistent rows known in column j. The situation occurs with probability at most $(1 - p)^{d(i)-1}$ since the jth column of any row is not seen with probability $(1 - p)$ and

this column must not be seen in each of the $d(i)-1$ other rows of the same type as row i. Combining these facts, we have

$$\begin{aligned} V_t &\leq \frac{1}{S}\sum_{i=1}^{n}\sum_{j=1}^{m}\left((1-p)^{d(i)-1}+\sum_{i'\neq i}\frac{h(i,i')p(1-p^2)^{h(i,i')-1}}{mp(d(i)\text{ - }\ )}\right) \\ &= \frac{1}{n}\sum_{i=1}^{n}(1-p)^{d(i)-1}+\frac{1}{S}\sum_{i=1}^{n}\sum_{i'\neq i}\frac{h(i,i')(1-p^2)^{h(i,i')-1}}{d(i)-1}. \end{aligned}$$

Recall that our goal is to upper bound the sum $\sum_{t=0}^{S-1} V_t$. Applying the above upper bound for V_t we get

$$\sum_{t=0}^{S-1} V_t \leq \sum_{t=0}^{S-1}\left(\frac{1}{n}\sum_{i=1}^{n}(1-p)^{d(i)-1}\right)+\sum_{t=0}^{S-1}\left(\frac{1}{S}\sum_{i=1}^{n}\sum_{i'\neq i}\frac{h(i,i')}{d(i)-1}(1-p^2)^{h(i,i')-1}\right). \tag{3}$$

We now evaluate the first part of the above expression. We begin by noting that

$$\sum_{t=0}^{S-1}\left(\frac{1}{n}\sum_{i=1}^{n}(1-p)^{d(i)-1}\right) \leq \frac{1}{n}\sum_{i=1}^{n}\left(1+\int_0^S\left(1-\frac{t}{S}\right)^{d(i)-1}dt\right).$$

Since

$$\int_0^S\left(1-\frac{t}{S}\right)^{d(i)-1}dt = \int_0^S\left(\frac{S-t}{S}\right)^{d(i)-1}dt = \int_0^S\left(\frac{t}{S}\right)^{d(i)-1}dt = \frac{S}{d(i)},$$

we get that

$$\begin{aligned} \frac{1}{n}\sum_{i=1}^{n}\left(1+\int_0^S\left(1-\frac{t}{S}\right)^{d(i)-1}dt\right) &= \frac{1}{n}\sum_{i=1}^{n}\left(1+\frac{S}{d(i)}\right) \\ &= 1+m\sum_{i=1}^{n}\frac{1}{d(i)} = km+1. \end{aligned} \tag{4}$$

We obtain the last step of the equality by rewriting the summation to go over all the row types: there are $n(r)$ terms for rows of type r and thus each row type contributes 1 to the summation.

We are now ready to evaluate the second part of Expression 3. To complete the proof of the theorem we must show that

$$\sum_{t=0}^{S-1}\left(\frac{1}{S}\sum_{i=1}^{n}\sum_{i'\neq i}\frac{h(i,i')}{d(i)-1}(1-p^2)^{h(i,i')-1}\right) = O(nk\sqrt{H}).$$

First we consider the case in which $h(i,i')=1$. In this case the second part of Expression simplifies to:

$$\sum_{i=1}^{n}\sum_{i'\neq i}\frac{1}{d(i)-1} = O(nk). \tag{5}$$

Next we consider when $h(i,i') > 1$. We begin by deriving an upper bound for the second part of Expression 3 of

$$\frac{1}{S}\sum_{i=1}^{n}\sum_{i'\neq i}\frac{h(i,i')}{d(i)-1}\left(1+\int_0^S\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}dt\right).$$

By analyzing the integral and applying the inequality $e^x \geq 1+x$ we get

$$\int_0^S\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}dt \leq \int_0^S \exp\left\{-\left(\frac{t}{S}\right)^2(h(i,i')-1)\right\}dt. \tag{6}$$

A standard integral table [9] gives

$$\int_0^\infty \exp\left\{-a^2x^2\right\}dx = \frac{1}{2a}\sqrt{\pi}.$$

So letting $a = \frac{\sqrt{h(i,i')-1}}{S}$ and $x = t$ one gets that

$$\int_0^\infty \exp\left\{-\left(\frac{t}{S}\right)^2(h(i,i')-1)\right\}dt = \frac{S\sqrt{\pi}}{2\sqrt{h(i,i')-1}}. \tag{7}$$

By combining equations (6) and (7) we get that

$$\int_0^S\left(1-\left(\frac{t}{S}\right)^2\right)^{h(i,i')-1}dt \leq \frac{S\sqrt{\pi}}{2\sqrt{h(i,i')-1}}. \tag{8}$$

Therefore, the second part of Expression 3 can be upper bounded by

$$\begin{aligned}
&\frac{1}{S}\sum_{i=1}^{n}\sum_{i'\neq i}\frac{h(i,i')}{d(i)-1}\left(1+\int_0^S\left(1-p^2\right)^{h(i,i')-1}dt\right)\\
&\qquad\leq \frac{1}{S}\sum_{i=1}^{n}\sum_{i'\neq i}\frac{h(i,i')}{d(i)-1}\left(1+\frac{S\sqrt{\pi}}{2\sqrt{h(i,i')-1}}\right)\\
&\qquad\leq \frac{m}{S}\sum_{i=1}^{n}\sum_{i'\neq i}\frac{1}{d(i)-1}+\frac{\sqrt{\pi}}{2}\sum_{i=1}^{n}\sum_{i'\neq i}\frac{h(i,i')}{(d(i)-1)\sqrt{h(i,i')-1}}\\
&\qquad= O\left(\frac{1}{n}nk+nk\sqrt{H}\right)=O(nk\sqrt{H}).
\end{aligned} \tag{9}$$

Finally putting together Equations 4, 5 and 9 we obtain an $O(km+nk\sqrt{H})$ bound as desired.

■

To complete the theorem, we prove the main result of this section, namely, the upper bound obtained under this simple probabilistic model holds (to within a constant factor) for the true model. In other words, to compute an upper bound on the number of mistakes made by a prediction algorithm when the instances are selected according to a uniform distribution on the instance space, one can replace the requirement that exactly t matrix entries are known by the requirement that each matrix entry is known with probability $\frac{t}{nm}$.

Lemma 5 $\sum_{t=0}^{S-1} U_t = O\left(\sum_{t=0}^{S-1} V_t\right)$.

Proof: We first note that

$$V_t = \sum_{r=0}^{S-1} \binom{S}{r} \left(\frac{t}{S}\right)^r \left(1 - \frac{t}{S}\right)^{S-r} U_r.$$

To see this, observe that for each r, where r is the number of known entries, we need just multiply U_r by the expectation that exactly r entries are known where each entry is known with probability of t/S. Therefore,

$$\begin{aligned} \sum_{t=0}^{S-1} V_t &= \sum_{t=0}^{S-1} \sum_{r=0}^{S-1} U_r \binom{S}{r} \left(\frac{t}{S}\right)^r \left(1 - \frac{t}{S}\right)^{S-r} \\ &= \sum_{r=0}^{S-1} U_r \left[\sum_{t=0}^{S-1} \binom{S}{r} \left(\frac{t}{S}\right)^r \left(1 - \frac{t}{S}\right)^{S-r}\right]. \end{aligned} \tag{10}$$

Thus, to prove the lemma, it suffices to show that the inner summation is bounded below by a positive constant. By symmetry, assume that $r \leq S/2$ and let $y = S - r$. By applying Stirling's approximation we obtain that

$$\binom{S}{r} = \Theta\left(\frac{S^S}{r^r y^y} \sqrt{\frac{S}{ry}}\right).$$

Applying this formula to the desired summation we obtain that

$$\begin{aligned} \sum_{t=0}^{S} \binom{S}{r} \left(\frac{t}{S}\right)^r \left(1 - \frac{t}{S}\right)^{S-r} &= \Theta\left(\sqrt{\frac{S}{ry}} \sum_{t=0}^{S} \left(\frac{t}{r}\right)^r \left(\frac{S-t}{y}\right)^y\right) \\ &= \Omega\left(\sqrt{\frac{S}{ry}} \sum_{x=1}^{\sqrt{ry/S}} \left(\frac{r+x}{r}\right)^r \left(\frac{y-x}{y}\right)^y\right). \end{aligned}$$

The last step above follows by letting $x = t - r$ and reducing the limits of the summation. To complete the proof that the inner summation of Equation 11 is bounded below by a positive constant we need just prove that

$$\left(\frac{r+x}{r}\right)^r \left(\frac{y-x}{y}\right)^y = \Omega(1)$$

for all $1 \leq x \leq \sqrt{ry/S}$

Using the inequality $1 + x \leq e^x$, it can be shown that for $1 + y \geq 0$, $1 + y \geq e^{\frac{y}{1+y}}$. We apply this observation to get that

$$\left(\frac{r+x}{r}\right)^r \left(\frac{y-x}{y}\right)^y = \left(1 + \frac{x}{r}\right)^r \left(1 - \frac{x}{y}\right)^y$$

$$\geq \exp\left\{\frac{x}{1+\frac{x}{r}} - \frac{x}{1-\frac{x}{y}}\right\} = \exp\left\{\frac{rx}{r+x} - \frac{yx}{y-x}\right\}$$
$$= \exp\left\{\frac{-x^2(r+y)}{(r+x)(y-x)}\right\} = \exp\left\{\frac{-Sx^2}{(r+x)(y-x)}\right\}.$$

Since $x \leq \sqrt{ry/S}$, it follows that $Sx^2 \leq ry$. Applying this observation to the above inequality we get that

$$\left(\frac{r+x}{r}\right)^r \left(\frac{y-x}{y}\right)^y \geq \exp\left\{\frac{-ry}{(r+x)(y-x)}\right\}$$
$$= \exp\left\{\frac{-ry}{ry+(y-r)x-x^2}\right\}$$
$$\geq \exp\left\{\frac{-ry}{ry-ry/S}\right\} = \exp\left\{\frac{-1}{1-\frac{1}{S}}\right\}.$$

Finally we note that for $S \geq 2$, $e^{\frac{-1}{1-1/S}} \geq e^{-2}$. This completes the proof of the lemma. ■

Combining Lemma 4 and Lemma 5 we obtain that $\sum_{t=0}^{S-1} U_t = O(km + nk\sqrt{H})$ giving the desired result. ■

This completes our discussion of learning a k-binary-relation.

4 Learning a Total Order

In this section we present our results for learning a binary relation on a set where it is known a priori that the relation forms a total order. One can view this problem as that of learning a total order on a set of n objects where an instance corresponds to comparing which of two objects is greater in the target total order. Thus this problem is like comparison-based sorting except for two key differences: we vary the agent selecting the order in which comparisons are made (in sorting the learner does the selection) and we charge the learner only for all *incorrectly predicted* comparisons.

Before describing our results, we motivate this section with the following example. There are n basketball teams that are competing in a round-robin tournament. That is, each team will play all other teams exactly once. Furthermore, we make the (admittedly simplistic) assumption that there is a ranking of the teams such that a team wins its match if and only if its opponent is ranked below it. The learner wants to place a \$10 bet on each game: if it bets on the winning team it wins \$10 and if it bets on the losing team it loses \$10. Of course, the goal of the learner is to win as many bets as possible. We formalize the problem of learning

a total order as follows. The instance space $X = \cup_{n \geq 1} X_n = \{1, \ldots, n\} \times \{1, \ldots, n\}$. An instance (i, j) in X is in the target concept if and only if object i precedes object j in the corresponding total order.

In the next section we discuss the relation between the halving algorithm and approximate counting. Then we show how to use an approximation scheme to implement a randomized version of the approximate halving algorithm, and apply this result to the problem of learning a total order on a set of n element. Finally, we discuss how a majority algorithm can be used to implement a counting algorithm.

4.1 The Halving Algorithm and Approximate Counting

In this section we review the halving algorithm and approximate counting schemes. We first cover the halving algorithm [3, 15]. Let $\mathcal{V}$ denote the set of concepts in C_n that are consistent with the feedback from *all previous queries*. Given an instance x in X_n, for each concept in $\mathcal{V}$ the halving algorithm computes the prediction of that concept for x and predicts according to the majority. Finally, all concepts in $\mathcal{V}$ that are inconsistent with the correct prediction are deleted. Littlestone [15] shows that this algorithm makes at most $\lg |C_n|$ mistakes. Now suppose the prediction algorithm predicts according to the majority of concepts in set $\mathcal{V}'$, the set of all concepts in C_n consistent with all *incorrectly* predicted instances. Littlestone [15] also proves that this *space-efficient halving algorithm* makes at most $\lg |C_n|$ mistakes. So for any prediction algorithm A that only remembers its mistakes, the number of instances stored by A is bounded by $\mathrm{MB}(A, C_n)$.

We define an *approximate halving algorithm* to be the following generalization of the halving algorithm. Given instance x in X_n an approximate halving algorithm predicts in agreement with at least $\varphi|\mathcal{V}|$ of the concepts in $\mathcal{V}$ for some constant $0 < \varphi < 1/2$.

Theorem 11 *An approximate halving algorithm makes at most* $\log_{(1-\varphi)^{-1}} |C_n|$ *mistakes for concept class* C.

Proof: Each time a mistake is made, the number of concepts that remain in $\mathcal{V}$ are reduced by a factor of at least $1-\varphi$. Thus after at most $\log_{(1-\varphi)^{-1}} |C_n|$ mistakes there is one consistent concept left in C_n. ∎

We note that the above result holds for the space-efficient version of the approximate halving algorithm.

We now introduce the notion of an approximate counting scheme for counting the number of elements in a finite set $\mathcal{S}$. Let x be a description of a set $\mathcal{S}_x$ in some natural encoding. An *exact counting scheme* on input x outputs $|\mathcal{S}_x|$ with probability 1. Such a scheme is polynomial if it runs in time polynomial in $|x|$. Sometimes exact counting can be done in polynomial time; however, the counting problem is often $\#\mathcal{P}$-complete and thus assumed to be intractable. (For a discussion of the class $\#\mathcal{P}$ see Valiant [22].) For many $\#\mathcal{P}$-complete problems good approximations are possible [12, 20, 21]. A *randomized approximation scheme*, R, for a counting problem satisfies the following condition for all $\epsilon, \delta > 0$:

$$\Pr\left[\frac{|\mathcal{S}_x|}{(1+\epsilon)} \leq R(x,\epsilon,\delta) \leq |\mathcal{S}_x|(1+\epsilon)\right] \geq 1-\delta$$

where $R(x,\epsilon,\delta)$ is R's estimate on input x, ϵ, and δ. Such a scheme is *fully polynomial* if it runs in time polynomial in $|x|, \frac{1}{\epsilon}$, and $\lg \frac{1}{\delta}$. For a further discussion see Sinclair [20].

We now review work on counting the number of linear extensions of a total order. That is, given a partial order on a set of n elements, the goal is to compute the number of total orders that are linear extensions of the given partial order. We discuss the relationship between this problem and that of computing the volume of a convex polyhedron. (For more details on this subject, see Section 2.4 of Lovász [17].) Given a convex set S and an element a of $\Re^n$ a weak separation oracle

1. Asserts that $a \in S$, or
2. Asserts that $a \notin S$ and supplies a reason why. In particular for closed convex sets in $\Re^n$, if $a \notin S$ then there exists a hyperplane separating a from S. So if $a \notin S$, the oracle responds with such a separating hyperplane as the reason why $a \notin S$.

We now discuss how to reduce the problem of counting the number of extensions of a partial order on n elements to that of computing the volume of a convex n-dimensional polyhedron given by a separation oracle. The polyhedron built in the reduction will be a subset of the unit hypercube in $\Re^n$ where the polyhedron is determined by the intersection of the halfspaces given by the inequalities of the partial order. (See Figure 9 for an example with $n = 3$.) For a total order the polyhedron is a simplex such that for any pair of total orders the simplices only intersect in a face (zero volume). Let T_n be the set of all $n!$ total orders on n elements. Then

$$\text{unit hypercube in } \Re^n = \bigcup_{t \in T_n} \text{polyhedron defined by } t. \tag{11}$$

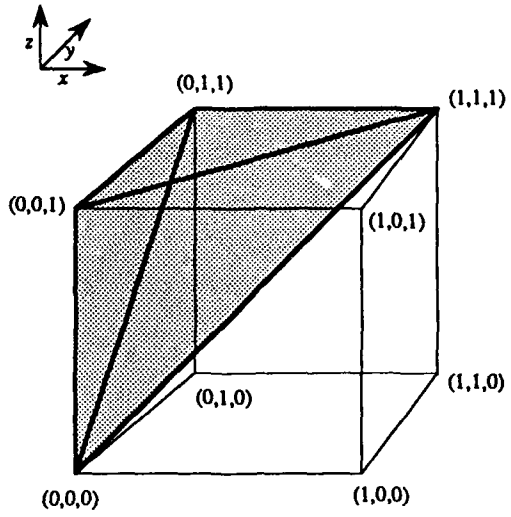

Figure 9: The polyhedron formed by the total order $z > y > x$.

Let P be a partial order on a set of n elements. From Equation 11 and the observation that the volumes of the polyhedra formed by each total order is equal, it follows that the volume of the polyhedron defined by any total order is $1/n!$. Thus it follows that for any partial order P

$$\frac{\text{number of extensions of } P}{n!} = \text{volume of polyhedron defined by } P. \tag{12}$$

Rewriting equation (12), we obtain that

$$\text{number of extensions of } P = n! \cdot (\text{volume of polyhedron defined by } P). \tag{13}$$

Finally, we note that the weak separation oracle is easy to implement for any partial order. Given inputs a and S, it just checks each inequality of the partial order to see see if a is in the convex polyhedron S. If a does not satisfy some inequality then reply that $a \notin S$ and return that inequality as the separating hyperplane. Otherwise, if a satisfies all inequalities, reply that $a \in S$.

Dyer, Frieze and Kannan [7] give a fully-polynomial randomized approximation scheme (*fpras*) to approximate the volume of a polyhedron given a separation oracle. From Equation 13 we see that this fpras for estimating the volume of a polyhedron can be easily applied to estimate the number of extensions of a partial order. Furthermore, Dyer and Frieze [8]

prove that it is $\#\mathcal{P}$-hard to exactly compute the volume a polyhedron given either by a list of its facets or its vertices.

Independently, Matthews [18] has described an algorithm to generate a random linear extension of a partial order. Consider the convex polyhedron K defined by the partial order. Matthew's main result is a technique to sample nearly uniformly from K. Given such a procedure to sample uniformly from K, one can sample uniformly from the set of extensions of a partial order by choosing a random point in K and then selecting the total order corresponding to the ordering of the coordinates of the selected point. A procedure to generate a random linear extension of a partial order can be used repeatedly to approximate the number of linear extensions of a partial order [18].

4.2 Application to Learning

In this section we show how to use a fpras to implement a randomized version of the approximate halving algorithm, and apply this result for the problem of learning a total order on a set of n elements.

Under the teacher-selected query sequence we obtain an $n-1$ mistake bound. The teacher can uniquely specify the target total order by giving the $n-1$ instances that correspond to consecutive elements in the target total order. Since $n-1$ instances are needed to uniquely specify a total order, we get a matching lower bound. Winkler [25] has shown that under the learner-selected query sequence, one can also obtain an $n-1$ mistake bound. To achieve this bound the learner uses an insertion sort, as described by Cormen, Leiserson, and Rivest [6], where for each new element the learner guesses it is smaller than each of the ordered elements (starting with the largest) until a mistake is made. When a mistake occurs this new element is properly positioned in the chain. Thus at most $n-1$ mistakes will be made by the learner. We now argue that learner can be forced to make $n-1$ mistakes. The adversary that gives feedback using the following simple strategy: the first time an object is involved in a comparison reply that learner's prediction is wrong. In doing so, one creates a set of *chains* where a chain is a total order on a subset of the elements. If c chains are created by this process then the learner has made $n-c$ mistakes. Since all these chains must be combined to get a total order, the adversary can force $c-1$ additional mistakes by always replying that a mistake occurs the first time when elements from two different chains are compared. (It is not hard to see that the above steps can be interleaved.) Thus the adversary can force

$n-1$ mistakes.

Next we consider the case that an adversary selects the query sequence. We first prove an $\Omega(n \lg n)$ lower bound on the number of mistakes made by any prediction algorithm. We use the following result of Kahn and Saks [13]. Given any partial order P that is not a total order there exists an incomparable pair of elements x_i,x_j such that

$$\frac{3}{11} \leq \frac{\text{number of extensions of } P \text{ with } x_i \leq x_j}{\text{number of extensions of } P} \leq \frac{8}{11}.$$

So the adversary can always pick a pair of elements so that regardless of the learner's prediction, the adversary can report that a mistake was made while only eliminating a constant fraction of the remaining total orders.

We now present a polynomial prediction algorithm making $n \lg n + (\lg e) \lg n$ mistakes with very high probability. We first show how to use an exact counting algorithm R, for counting the number of concepts in C_n consistent with a given set of examples, to implement the halving algorithm.

Lemma 6 *Given a polynomial algorithm R to exactly count the number of concepts in C_n consistent with a given set E of examples, one can construct an efficient implementation of the halving algorithm for C.*

Proof: We show how to use R to efficiently make the predictions required by the halving algorithm. To make a prediction for an instance x in X_n the following procedure is used: Construct E^- from E by appending x as a negative example to E. Use the counting algorithm R to count the number of concepts $C^- \in \mathcal{V}$ that are consistent with E^-. Next construct E^+ from E by appending x as a positive example to E. As before, use R to count the number of concepts $C^+ \in \mathcal{V}$ that are consistent with E^+. Finally if $C^- \geq C^+$ then predict that x is a negative example, otherwise predict that x is a positive example.

Clearly a prediction is made in polynomial time, since it just requires calling R twice. We claim that it predicts according to the majority of concepts in $\mathcal{V}$. Note that C^- is the number of concepts in $\mathcal{V}$ for which x is a negative instance. Likewise, C^+ is the number of concepts in $\mathcal{V}$ for which x is a positive instance. Thus it immediately follows that the prediction agrees with the majority of concepts in $\mathcal{V}$. ∎

We modify this basic technique to use a fpras instead of the exact counting algorithm to obtain an efficient implementation of a randomized version of the approximate halving algorithm.

Theorem 12 *Let R be a* fpras *for counting the number of concepts in $\mathcal{C}_n$ consistent with a given set E of examples. If $|X_n|$ is polynomial in n, one can produce a prediction algorithm that for any $\delta > 0$ runs in time polynomial in n and in $\lg \frac{1}{\delta}$ and makes at most $\lg |C_n| \left(1 + \frac{\lg e}{n}\right)$ mistakes with probability at least $1 - \delta$.*

Proof: The prediction algorithm implements the procedure described in Lemma 6 with the exact counting algorithm replaced by the fpras $R(n, \frac{1}{n}, \frac{\delta}{2|X_n|})$. Consider the prediction for an instance $x \in X_n$. Let $\mathcal{V}$ be the set of concepts that are consistent with all previous instances. Let r^+ (respectively r^-) be the number of concepts in $\mathcal{V}$ for which x is a positive (negative) instance. Let $\hat{r}^+$ (respectively $\hat{r}^-$) be the estimate output by R for r^+ (r^-). Since R is a fpras, with probability at least $1 - \frac{\delta}{|X_n|}$

$$\frac{r^-}{1+\epsilon} \leq \hat{r}^- \leq (1+\epsilon) r^- \quad \text{and} \quad \frac{r^+}{1+\epsilon} \leq \hat{r}^+ \leq (1+\epsilon) r^+.$$

Without loss of generality, assume that the algorithm predicts that x is a negative instance, and thus $\hat{r}^- \geq \hat{r}^+$. Combining the above inequalities and the observation that $r^- + r^+ = |V|$, we obtain that $r^- \geq \frac{|V|}{1+(1+\epsilon)^2}$.

We define an *appropriate* prediction to be a prediction that agrees with *at least* $\frac{|V|}{1+(1+\epsilon)^2}$ of the concepts in V. To analyze the mistake bound for this algorithm, suppose that each prediction is appropriate. For a single prediction to be appropriate, both calls to the fpras R must output a count that is within a factor of $1+\epsilon$ of the true count. So any given prediction is appropriate with probability at least $1 - \frac{\delta}{|X_n|}$, and thus the probability that all predictions are appropriate is at least

$$1 - |X_n| \left(\frac{\delta}{|X_n|}\right) = 1 - \delta.$$

Clearly if all predictions are appropriate then the above procedure is in fact an implementation of the approximate halving algorithm with $\varphi = \frac{1}{1+(1+\epsilon)^2}$ and thus by Theorem 11 at most $\log_{(1-\varphi)^{-1}} |C_n|$ mistakes are made. Substituting ϵ with its value of $\frac{1}{n}$ and simplifying the expression we obtain that with probability at least $1 - \delta$,

$$\#\text{ mistakes} \leq \frac{\lg |C_n|}{\lg \frac{1}{1-\varphi}} = \frac{\lg |C_n|}{\lg \left(1 + \frac{n^2}{n^2+2n+1}\right)}. \tag{14}$$

Since $\frac{n^2}{n^2+2n+1} \geq 1 - \frac{2}{n}$,

$$\frac{1}{\lg \left(1 + \frac{n^2}{n^2+2n+1}\right)} \leq \frac{1}{\lg \left(1 + 1 - \frac{2}{n}\right)}$$

$$= \frac{1}{1+\lg\left(1-\frac{1}{n}\right)}$$
$$= 1-\frac{\lg\left(1-\frac{1}{n}\right)}{1+\lg\left(1-\frac{1}{n}\right)}$$

Applying the inequalities $\lg\left(1-\frac{1}{n}\right) \geq \frac{-\lg e}{n-1}$ and $1+\lg\left(1-\frac{1}{n}\right) \leq 1-\frac{\lg e}{n}$ we get that

$$\frac{\lg\left(1-\frac{1}{n}\right)}{1+\lg\left(1-\frac{1}{n}\right)} \geq \frac{\frac{-\lg e}{n-1}}{1-\frac{\lg e}{n}}$$
$$= \frac{-\lg e}{n-1-\frac{n-1}{n}\lg e}$$
$$\geq \frac{-\lg e}{n}$$

Finally, applying these inequalities to Equation 14 yields that

$$\#\text{ mistakes} \leq \frac{\lg|C_n|}{\lg\left(1+\frac{n^2}{n^2+2n+1}\right)} \leq \lg|C_n|\left(1+\frac{\lg e}{n}\right).$$

■

Note that we could modify the above proof by not requiring that all predictions be appropriate. In particular if we allow γ predictions to not be appropriate then we get a mistake bound of $\lg|C_n|\left(1+\frac{\lg e}{n}\right)+\gamma$.

We now apply this result to obtain the main result of this section. Namely, we describe a randomized polynomial prediction algorithm for learning a total order in the case that the adversary selects the query sequence.

Theorem 13 *There exists a prediction algorithm A for learning total orders such that on input δ (for all $\delta > 0$), and for any query sequence provided by the adversary, A runs in time polynomial in n and $\lg\frac{1}{\delta}$ and makes at most $n\lg n+(\lg e)\lg n$ mistakes with probability at least $1-\delta$.*

Proof Sketch: We apply the results of Theorem 12 using the fpras for counting the number of extensions of a partial order given independently by Dyer, Frieze and Kannan [7], and by Matthews [18]. We know that with probability at least $1-\delta$, the number of mistakes is at most $\lg|C_n|\left(1+\frac{\lg e}{n}\right)$. Since $|C_n| = n!$ the desired result is obtained. ■

We note that the probability that A makes more than $n\lg n+(\lg e)\lg n$ mistakes does not depend on the query sequence selected by the adversary. The probability is taken over the coin flips of the randomized approximation scheme.

Thus, as in learning a k-binary-relation using a row-filter algorithm, we see that a learner can do asymptotically better with self-directed learning versus adversary-directed learning. However, while the self-directed learning algorithm is deterministic, the adversary-directed algorithm is randomized. To accommodate such randomized prediction algorithms we provide the learner with an input δ and allow the algorithm to exceed the "worst-case" mistake bound with a probability δ.

4.3 Majority Algorithms vs Counting Algorithms

In this section we discuss the relationship between counting schemes and the halving algorithm. Let $\mathcal{W}$ be a set of elements for which some subset $\mathcal{S}$ of the elements are distinguished. We use the function g that maps an element of $\mathcal{W}$ to $\{0,1\}$ to represent which of the elements of W are distinguished. Specifically, for $w \in \mathcal{W}$, $g(w) = 1$ if and only if w is a distinguished element. Let Σ^* be an alphabet used to describe some subset of $\mathcal{W}$. Let f be a function from $\Sigma^* \rightarrow 2^{\mathcal{W}}$ that maps $\sigma \in \Sigma^*$ to the subset of $\mathcal{W}$ that it describes. We denote $f(\sigma)$ by $\mathcal{V}_\sigma$. Let $t_\sigma = |\mathcal{V}_\sigma|$, and let d_σ be the number of distinguished elements in $\mathcal{V}_\sigma$. Formally, we have $d_\sigma = |\{w \in \mathcal{V}_\sigma : g(w) = 1\}|$. Finally, $u_\sigma = |\mathcal{V}_\sigma| - d_\sigma$. That is, u_σ is the number of undistinguished elements in $\mathcal{V}_\sigma$. So $d_\sigma + u_\sigma = t_\sigma$.

A *majority algorithm* takes as input σ and outputs a bit that is 1 if $d_\sigma \geq u_\sigma$, and 0 if $d_\sigma \leq u_\sigma$. That is,

$$\text{MAAJORITY}(\sigma) = \begin{cases} 1 & \text{if } d_\sigma \geq u_\sigma \\ 0 & \text{if } d_\sigma \leq u_\sigma \end{cases}$$

On the other hand, a *counting algorithm* must output d_σ. In Lemma 6 we used a counting algorithm to implement a majority algorithm. (There $\mathcal{V}_\sigma = C_n$ and an element of $\mathcal{V}_\sigma$ is distinguished if and only if it is consistent with the given set of examples.) In this section we discuss using a majority algorithm to implement a counting algorithm. The results of this section are preliminary. Although we describe two techniques to convert a majority algorithm to a counting algorithm, we do not have applications for these techniques that yield any previously unknown results. We apply the first technique to an example problem; however, it is trivial to construct a counting algorithm for this problem.

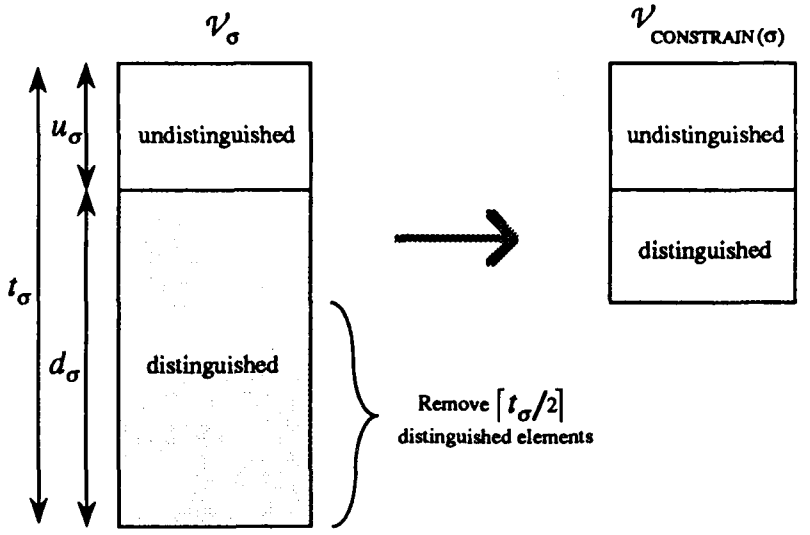

Figure 10: An illustration of the CONSTRAIN oracle when the majority of the elements in $\mathcal{V}_\sigma$ are distinguished.

4.3.1 First Approach

In this section we describe our first approach for using a majority algorithm to implement a counting algorithm. We then show that our algorithm can be used to convert an algorithm that outputs if a majority of k-CNF formulae classify a given instance as positive to one that computes the number of k-CNF formulae that classify the given instance as positive.

We now describe our first approach. Here we recursively apply MAJORITY, at each step reducing the larger of d_σ and u_σ by a factor of at least two. To use this approach, in addition to the MAJORITY oracle, a CONSTRAIN oracle must be provided. The specification for the CONSTRAIN oracle is as follows:

$$\text{CONSTRAIN}(\sigma) = \begin{cases} \text{new–object}(d_\sigma - \lceil t_\sigma/2 \rceil, u_\sigma) & \text{if } d_\sigma \geq u_\sigma \text{ (i.e. MAAJORITY}(\sigma) = 1) \\ \text{new–object}(d_\sigma, u_\sigma - \lceil t_\sigma/2 \rceil) & \text{if } d_\sigma \leq u_\sigma \text{ (i.e. MAAJORITY}(\sigma) = 0) \end{cases}$$

where new–object(d, u) creates a word $\sigma \in \Sigma^*$ such that $d_\sigma = d$ and $u_\sigma = u$. So the oracle CONSTRAIN just reduces the larger of d_σ and u_σ by a factor of $\lceil t_\sigma/2 \rceil$. The result of the CONSTRAIN oracle is illustrated in Figure 10.

Theorem 14 *One can construct, from a* MAJORITY *and* CONSTRAIN *oracle, a counting algorithm that on input* σ *uses at most* $\lg(t_\sigma)$ *calls to both oracles.*

Proof: We construct recursive counting procedure that takes input σ, and uses CONSTRAIN to reduce the larger of d_σ and u_σ. The initial call should be *Exact-Count1*(σ).

Exact-Count1(σ)
1 **if** $t_\sigma = 0$
2 **then return** 0
3 **else return** $\lceil t_\sigma/2 \rceil$MAAJORITY(σ) + *Exact-Count1*(CONSTRAIN(σ), $\lfloor t_\sigma/2 \rfloor$)

We first argue that this procedure is correct. Each time CONSTRAIN is called, we know that d_σ is reduced by exactly $\lceil t_\sigma/2 \rceil$MAAJORITY(σ). So the total decrease in d_σ from its original value is accumulated in line 3 of the procedure. Finally, when $t_\sigma = 0$ then clearly $d_\sigma = 0$, thus the base case is correct.

Since at each step t_σ is reduced to $\lfloor t_\sigma/2 \rfloor$, at most $\lg(t_\sigma)$ recursive calls are made. Furthermore, MAJORITY and CONSTRAIN are called only once for each recursive call. ■

In terms of our original goal of converting a majority algorithm into a counting algorithm we have the following corollary of Theorem 14.

Corollary 3 *Let $\mathcal{M}$ be a majority algorithm for $\mathcal{W}$ using Σ^*. Let $T_{\mathcal{M}}(\sigma)$ be the running time of $\mathcal{M}$ on input σ. Furthermore, suppose that* CONSTRAIN *can be implemented in time $T_C(\sigma)$ on input σ. Then there exists an exact counting algorithm that runs in time at most $(T_{\mathcal{M}}(\sigma) + T_C(\sigma)) \lg t_\sigma$.*

We now apply this conversion to the following problem. Given a boolean vector $x = \{0,1\}^n$, compute the number of k-CNF formula over n variables for which x is a positive example. As Angluin notes [2], Valiant's algorithm for PAC-learning k-CNF implements the halving algorithm. That is, given an instance x, it replies that x is a positive instance if and only if at least half of the remaining k-CNF formulas classify x as a positive example. Furthermore, each prediction is made in polynomial time. Therefore, for the given counting problem, we have the needed majority algorithm. We now apply Corollary 3 to k-CNF to obtain the following result.

Lemma 7 *There exists a polynomial time algorithm to exactly count the number of k-CNF formulas for which some $x \in X_n$ is a positive instance.*

Proof: Let σ be select from $\{0,1\}^n$ where the interpretation is that σ gives the assignments to the n variables. So $\mathcal{V}_\sigma$ is computed by evaluating the target formula on the assignment

given by σ and including those that evaluate to 1. Thus Valiant's [23] algorithm for learning k-CNF can serve as the majority algorithm. We now prove that the CONSTRAIN oracle can also be implemented in polynomial time. As Angluin [2] notes, if Valiant's algorithm predicts 0, then there exists some clause τ in the learner's hypothesis that evaluates to 0. So by removing τ the requirements of CONSTRAIN are satisfied. ■

We note that this result is easily obtained without using Corollary 3. By studying the recursive structure of the counting algorithm, we obtain the following counting algorithm for k-CNF. Let T be the number of possible clauses of size k or less that are true for instance x. Since the target formula can contain any subset of these clauses (but none of the clauses that ate negative for x), the number of k-CNF formulas that predict that x is a positive instance is 2^T.

4.3.2 Second Approach

In this section we describe our second approach to using a majority algorithm to implement a counting algorithm. To motivate this approach, we consider how the first approach might fail. The algorithm *Exact-Count1* can be used only if one can remove elements of $\mathcal{V}_\sigma$ in a controlled manner. However, it may be the case that one cannot remove elements from $\mathcal{V}_\sigma$ as desired, but can create some σ' such that $\mathcal{V}_{\sigma'} \supset \mathcal{V}_\sigma$ and furthermore, all elements of $\mathcal{V}_{\sigma'} - \mathcal{V}_\sigma$ are distinguished (or undistinguished). Our second approach is based on these ideas; instead of adjusting the size of d_σ and u_σ by reducing their sizes, we achieve the same effect by appropriately increasing the size of the smaller one.

To use this approach, in addition to the MAJORITY oracle, a EXPAND oracle must be provided instead of the CONSTRAIN oracle. The specification of the EXPAND oracle is as follows:

$$\text{EXPAND}(\sigma, i) = \begin{cases} \text{new-object}(d_\sigma, u_\sigma + i) & \text{if } d_\sigma \geq u_\sigma \\ \text{new-object}(d_\sigma + i, u_\sigma) & \text{if } d_\sigma \leq u_\sigma \end{cases}$$

So the oracle EXPAND just adds i elements to the smaller of d_σ and u_σ. The result of the EXPAND oracle is illustrated in Figure 11.

We now describe the details of the second approach to convert a majority algorithm to a counting algorithm.

Theorem 15 *One can construct, from a* MAJORITY *and an* EXPAND *oracle, a counting algorithm that on input σ uses at most $\lg(t_\sigma)$ calls to both oracles.*

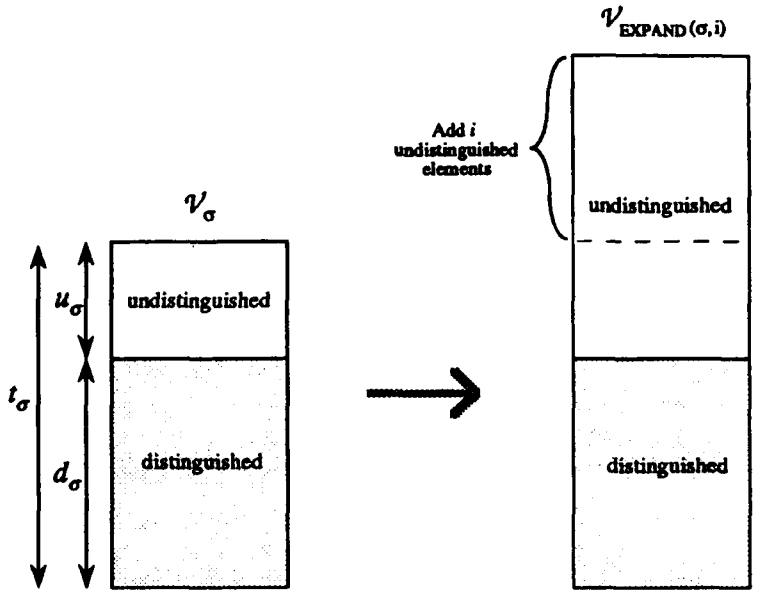

Figure 11: An illustration of the EXPAND oracle for the case that a majority of $\mathcal{V}_\sigma$ is distinguished.

Proof: As in Theorem 14, the basic idea is to use the MAJORITY oracle to compute d_σ. We do this by increasing the smaller of d_σ and u_σ so that they are essentially equally. As before, we use a binary search technique to efficiently perform this task, where at each step we use a call to MAJORITY to determine in which direction further adjustments should be made. The details are as follows.

Exact-Count2(σ, β, i)
1 **if** $i = 0$
2 **then if** MAAJORITY$(\sigma) = 1$
3 **then return** $(t_\sigma + \beta)/2$
4 **else return** $(t_\sigma - \beta)/2$
5 **else if** MAAJORITY(EXPAND$(\sigma, \beta)) =$ MAAJORITY(σ)
6 **then** *Exact-Count2*$(\sigma, \beta + \lceil i/2 \rceil, \lfloor i/2 \rfloor)$
7 **else** *Exact-Count2*$(\sigma, \beta - \lceil i/2 \rceil, \lfloor i/2 \rfloor)$

The initial call should be *Exact-Count2* $(\sigma, 0, t_\sigma)$. Note that β is the current estimate of the number of elements that need to be added to the small side to make both sides equal, and i is the size of the next adjustment to be made to β.

We first argue that the procedure is correct. Let β_l, i_l denote the input values of β and i

to the lth call of *Exact-Count2*. Let $\sigma_l = \text{EXPAND}(\sigma, \beta_i)$. Finally, we use d_l (respectively u_l) to denote d_{σ_l} (respectively u_{σ_l}). Without loss of generality, assume that $d_\sigma > u_\sigma$. So for all l,

- $d_l = d_0 = d_\sigma$,
- $u_l = u_0 + \beta_l = u_\sigma + \beta_l$, and
- $i_l = \lfloor i_{l-1}/2 \rfloor$.

We claim that when the input $i = 0$ in *Exact-Count2*, $d_l = u_l$. We prove this using the following lemma.

Lemma 8 *For all l, $|d_{\sigma_l} - u_{\sigma_l}| \leq i_l$.*

Proof: We use an inductive proof on l. Clearly, the base case, $|d_\sigma - u_\sigma| \leq t_\sigma$, holds.

We now prove the inductive step. Assume inductively that $|d_l - u_l| = |d_\sigma - u_\sigma - \beta_l| \leq i_l$. Our goal is to show that

$$|d_\sigma - u_\sigma - \beta_{l+1}| \leq i_{l+1} = \lfloor i_l/2 \rfloor.$$

We separately consider when $d_l > u_l$, $d_l < u_l$, and $d_l = u_l$.

- **Case 1:** $d_l > u_l$. In this case, $\beta_{l+1} = \beta_l + \lceil i_l/2 \rceil$. So we must show that

$$-\lfloor i_l/2 \rfloor \leq d_\sigma - u_\sigma - \beta_l - \lceil i_l/2 \rceil \leq \lfloor i_l/2 \rfloor.$$

  That is, we must show that:

$$\lceil i_l/2 \rceil - \lfloor i_l/2 \rfloor \leq d_\sigma - u_\sigma - \beta_l \leq \lfloor i_l/2 \rfloor + \lceil i_l/2 \rceil = i_l.$$

  The inequality $d_\sigma - u_\sigma - \beta_l \leq i_l$ follows immediately from the inductive hypothesis. For the other inequality note that $\lceil i_l/2 \rceil - \lfloor i_l/2 \rfloor \leq 1$. Furthermore, since $d_l > u_l$, $d_\sigma - u_\sigma - \beta_l = d_l - u_l > 1$.

- **Case 2:** $d_l < u_l$. In this case, $\beta_{l+1} = \beta_l - \lceil i_l/2 \rceil$. So we must show that

$$-\lfloor i_l/2 \rfloor \leq d_\sigma - u_\sigma + \beta_l - \lceil i_l/2 \rceil \leq \lfloor i_l/2 \rfloor.$$

  The proof for these inequalities is similar to that of Case 1.

- **Case 3:** $d_l = u_l$. For this case we first use an inductive proof to show that $d_l + u_l \equiv i_l \pmod 2$. The base case follows from the fact that $i_0 = d_0 + u_0$. For the inductive step, observe that $i_l - i_{l+1} = |(d_l + u_l) - (d_{l+1} + u_{l+1})|$. Since $d_l = u_l$, it must be that i_l divisible by two, and thus $\lfloor i_l/2 \rfloor = \lceil i_l/2 \rceil$. The inductive hypothesis immediately follows from here.

This completes the proof of the lemma. ∎

Since the recursion continues until $i_l = 0$ it follows from Lemma 8 that when $i = 0$ in *Exact-Count2*, $d_l = u_l$. Thus we know that for β the final value of β_l, we have that $d_\sigma = u_\sigma + \beta$. Applying the equality that $d_\sigma + u_\sigma = t_\sigma$ and solving for d_σ, we get that $d_\sigma = (t_\sigma + \beta)/2$. (Since $d_l = u_l$, it follows that $t_\sigma + \beta$ is even.) The case where $d_\sigma \leq u_\sigma$ is handled similarly. Thus we have shown that the output from *Exact-Count2* is correct.

Since at each step the increment size i is reduced to $\lfloor i/2 \rfloor$, at most $\lg(t_\sigma)$ recursive calls are made. Furthermore each oracle is called at most once during each recursive call. ∎

In terms of our original goal of converting a majority algorithm into a counting algorithm we have the following corollary of Theorem 15.

Corollary 4 *Let $\mathcal{M}$ be a majority algorithm for $\mathcal{W}$ under Σ^*. Let $T_{\mathcal{M}}$ be the running time of $\mathcal{M}$ on input σ. Furthermore, suppose that* EXPAND *can be implemented in time T_C on input of size at most t_σ. Then there exists an exact counting algorithm that runs in time at most $(T_{\mathcal{M}} + T_C) \lg t_\sigma$.*

5 Conclusions and Open Problems

We have studied the the problem of learning a binary relation between two sets of objects and between a set and itself under an extension of the on-line learning model. We have presented general techniques to help develop efficient versions of the halving algorithm. In particular, we have shown how a fpras can be used to efficiently implement a randomized version of the approximate halving algorithm. We have also extended the mistake bound model by adding the notion of an instance selector and generalizing it to accommodate randomized learning algorithms. The specific results are summarized in Table 1. In this table all lower bounds are information-theoretic bounds and all upper bounds are for polynomial-time learning algorithms. Also, unless otherwise stated, the results listed are for deterministic learning algorithms.

Concept Class	Director	Lower Bound	Upper Bound	Notes
Binary Relation (k row types)	Learner	$\frac{km}{2} + (n - \frac{k}{2})\lfloor \lg k - 1 \rfloor$	$km + (n-k)\lceil \lg k \rceil$	
	Teacher	$km + (n-k)(k-1)$	$km + (n-k)(k-1)$	
	Adversary	$km + (n-k)\lfloor \lg k \rfloor$	$O(km + n\sqrt{m \lg k})$ *	
	Adversary	$2m + n - 2$	$2m + n - 2$	$k = 2$
	Adversary	$\Omega(km + (n-k)\lg k + \min\{n\sqrt{m}, m\sqrt{n}\})$	$km + n\sqrt{(k-1)m}$	row-filter algorithm
	Uniform Dist.	$\frac{km}{2} + (n - \frac{k}{2})\lfloor \lg k - 1 \rfloor$	$O(km + nk\sqrt{H})$	avg. case, row-filter alg.
Total Order	Teacher	$n - 1$	$n - 1$	
	Learner	$n - 1$	$n - 1$ †	
	Adversary	$\Omega(n \lg n)$	$n \lg n + (\lg e) \lg n$	randomized algorithm

Table 1: Summary of our results.

From observing Table 1 one can see that several of the above bounds are tight and several others are asymptotically tight. However, for the problem of learning a k-binary-relation there is a gap in the bound for the random and adversary (except $k \leq 2$) directors. Note that the bounds for row-filter algorithms are asymptotically tight for k constant. Clearly, if we want asymptotically tight bounds that include a dependence on k we must incorporate k into the projective geometry lower bound. (Currently, the relation created by the adversary has only two row types.)

For the problem of learning a total order, all the above bounds are tight or asymptotically tight. Although the fpras for approximating the number of extensions of a partial order is a polynomial-time algorithm, the exponent on n is somewhat large and the algorithm is quite complicated. Thus an interesting problem is to find a "practical" prediction algorithm for the problem of learning a total order. Another interesting direction of research is to explore other ways of modeling the structure in a binary relation. Finally, we hope to find other applications of fully polynomial approximation schemes to learning theory.

*Due to Manfred Warmuth.

†Due to Peter Winkler.

Acknowledgments

The results in Sections 4.1 and 4 were inspired by an "open problems session" led by Manfred Warmuth at our weekly Machine Learning Reading Group meeting, where he proposed the basic idea of using an approximate halving algorithm based on approximate and probabilistic counting, and also suggested the problem of learning a total order on n elements. We thank Tom Leighton for helping to improve the lower bound of Theorem 9. We also thank Nick Littlestone, Bob Sloan, and Ken Goldman for their comments.

References

[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[2] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[3] J. Barzdin and R. Freivald. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, 13:1224–1228, 1972.

[4] Avrim Blum. An $\tilde{O}(n^0.4)$-approximation algorithm for 3-coloring. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 535–542, Seattle, Washington, May 1989.

[5] R. Carmichael. *Introduction to the Theory of Groups of Finite Order.* Dover Publications, New York, 1937.

[6] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction t· Algorithms.* MIT Press/McGraw-Hill, 1990.

[7] M. Dyer, A. Frieze, and R. Kannan. A random polynomial time algorithm for estimating the volumes of convex bodies. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 375–381, Seattle, Washington, May 1989.

[8] Martin Dyer and Alan Frieze. On the complexity of computing the volume of a polyhedron. *SIAM Journal on Computing*, 17:967–974, 1988.

[9] T. Gradshteyn and I. Ryzhik. *Tables of Integral, Series, and Products.* Academic Press, New York, 1980. corrected and enlarged edition by A. Jeffrey.

[10] David Haussler, Michael Kearns, Nick Littlestone, and Manfred K. Warmuth. Equivalence of models for polynomial learnability. In *First Workshop on Computational Learning Theory*, pages 42–55. Morgan-Kaufmann, August 1988.

[11] David Haussler, Nick Littlestone, and Manfred Warmuth. Expected mistake bounds for on-line learning algorithms. Unpublished manuscript, 1988.

[12] Mark Jerrum and Alistair Sinclair. Conductance and the rapid mixing property for Markov chains: the approximation of the permanent resolved. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 235–244, May 1988.

[13] Jeff Kahn and Michael Saks. Balancing poset extensions. *Order 1*, pages 113–126, 1984.

[14] Nati Linial and Umesh Vazirani. Graph products and chromatic numbers. In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 124–128, Research Triangle Park, North Carolina, October 1989.

[15] Nick Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[16] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 256–261, Research Triangle Park, North Carolina, October 1989.

[17] László Lovász. An algorithmic theory of numbers, graphs and convexity. In *CBMS-NSF Regional Conference Series on Applied Mathematics*, Philadelphia, 1986.

[18] Peter Matthews. Generating a random linear extension of a partial order. Unpublished manuscript, 1989.

[19] Ronald L. Rivest and Robert Sloan. Learning complicated concepts reliably and usefully. In David Haussler and Leonard Pitt, editors, *First Workshop on Computational Learning Theory*, pages 69–79. Morgan Kaufmann, August 1988.

[20] Alistair Sinclair. *Randomised Algorithms for Counting and Generating Combinatorial Structures.* PhD thesis, University of Edinburgh, Department of Computer Science, November 1988.

[21] Larry Stockmeyer. The complexity of approximate counting. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 118–126, May 1983.

[22] Leslie Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:198–201, 1979.

[23] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[24] Manfred Warmuth. Personal communication.

[25] Peter Winkler. Personal communication.

OFFICIAL DISTRIBUTION LIST

DIRECTOR — 2 copies
Information Processing Techniques Office
Defense Advanced Research Projects Agency (DARPA)
1400 Wilson Boulevard
Arlington, VA 22209

OFFICE OF NAVAL RESEARCH — 2 copies
800 North Quincy Street
Arlington, VA 22217
Attn: Dr. Gary Koop, Code 433

DIRECTOR, CODE 2627 — 6 copies
Naval Research Laboratory
Washington, DC 20375

DEFENSE TECHNICAL INFORMATION CENTER — 12 copies
Cameron Station
Alexandria, VA 22314

NATIONAL SCIENCE FOUNDATION — 2 copies
Office of Computing Activities
1800 G. Street, N.W.
Washington, DC 20550
Attn: Program Director

HEAD, CODE 38 — 1 copy
Research Department
Naval Weapons Center
China Lake, CA 93555